

Discovering the value of IBM Integration Bus v9 for your ESB and SOA

Lab Exercises



Contents

LAB 1	GETTING STARTED	5
1.1	BUILDING AND EXECUTING A SIMPLE MESSAGE FLOW.....	5
1.2	GETTING TO KNOW THE TOOLKIT	6
1.3	BUILDING A SIMPLE FLOW.....	10
1.4	TESTING THE FLOW	25
LAB 2	MESSAGE MODELS AND WORKING WITH XML MESSAGES.....	41
2.1	OVERVIEW	41
2.2	USING THE XML PARSER	41
2.3	CREATING A MESSAGE MODEL FROM AN XSD	47
2.4	RE-RUNNING THE TEST CLIENT	61
LAB 3	CONTENT-BASED ROUTING AND THE DEBUGGER	63
3.1	OVERVIEW	63
3.2	ADD ROUTING LOGIC	63
3.3	TEST WITH THE DEBUGGER	80
3.4	A CLOSER LOOK AT THE DEPLOYMENT PROCESS.....	91
LAB 4	BUILDING A WEB SERVICE USING PATTERNS AND MAPPING	99
4.1	LAB OVERVIEW	99
4.2	TEST THE MESSAGE FLOW	142
LAB 5	USING .NET IN IBM INTEGRATION BUS	156
5.1	OVERVIEW	156
5.2	INSTALL THE SAMPLE	156
5.3	CREATE THE .NET CLASSES	163
5.4	CREATE THE CONFIGURABLE SERVICE	181
5.5	DEPLOY AND EXECUTE THE MESSAGE FLOW.....	189
5.6	CLEANUP	201
LAB 6	WORKING WITH MOBILE APPLICATIONS.....	202
6.1	OVERVIEW	202
6.2	MICROSOFT .NET REQUEST/RESPONSE MOBILE PATTERN.....	203
6.3	DEPLOY THE WORKLIGHT ADAPTERS AND TEST THE APPLICATION	220
6.4	TEST THE ADAPTER USING THE ANDROID EMULATOR.....	242
6.5	THE RESOURCE HANDLER MOBILE PATTERN	270
6.6	INSTALL AND TEST THE MOBILECARS ADAPTER	282
6.7	CLEAN UP	309
LAB 7	WORKING WITH FILES	310
7.1	OVERVIEW	310
7.2	CREATE THE MESSAGE FLOW	311
7.3	EXECUTE THE MESSAGE FLOW	322
7.4	TEST WITH A VERY LARGE FILE	335
7.5	CLEAN UP	343
LAB 8	DFDL AND MESSAGE MODEL TOOLING.....	344
8.1	INTRODUCTION TO MESSAGE MODEL STANDARDS	344
8.2	CREATING A CSV MESSAGE MODEL.....	345
8.3	TEST THE MESSAGE MODEL.....	372
APPENDIX A.	NOTICES	384
APPENDIX B.	TRADEMARKS AND COPYRIGHTS	386

THIS PAGE INTENTIONALLY LEFT BLANK

Lab 1 Getting started

1.1 Building and executing a simple message flow

In this lab, you will build and execute a simple message flow. A message flow is similar to a program but is developed using a visual paradigm.

The flow will be deployed to an integration server in an integration node where it will execute. An integration server is an operating system process where user flows execute. The flow will then be available to process messages. There is no need to restart the integration node or the integration server for the deployment of a new or changed message flow.

The unit of deployment is a broker archive file (BAR). Broker archive files have a file extension of “bar”. It is essentially a zip file with a deployment descriptor. The deployment descriptor allows certain properties of the message flow to be overridden.

The broker archive file will hold the artifacts to be deployed to a specific integration server in a specific integration node. It may contain applications and libraries as well as message flows, message sets, XSL Transformations (XSLT) style sheets, Java™ Archive (JAR) files, XML schema files or WSDL files. In addition, the related source files might also be included. When you add a message flow to the BAR file, additional validation of the message flow is performed. The BAR file is then deployed to the integration node. The final validation of the artifacts is done by the integration node. If errors are found by the integration node, they will be reported back in the event log.

As a convention for these labs, a red box will be used to identify a particular area, and when information is to be entered or an action is to be taken, it will be in **bold** characters. Red lines might be used to indicate where nodes are to be placed when building your message flow.

Icons are provided on the desktop. These are:

- Integration toolkit
- Integration console
- Integration Bus Explorer (IBM® WebSphere® MQ Explorer)

The system is Windows 7-X64. The IBM software includes IBM WebSphere MQ Version 7.5, IBM Integration Bus V9, Worklight Studio V5.0.6 and IBM DB2® V9.7. The Worklight Studio V5.0.6 includes the Android SDK and Eclipse plug-in. Other IBM software is also installed and will be described in the later labs.

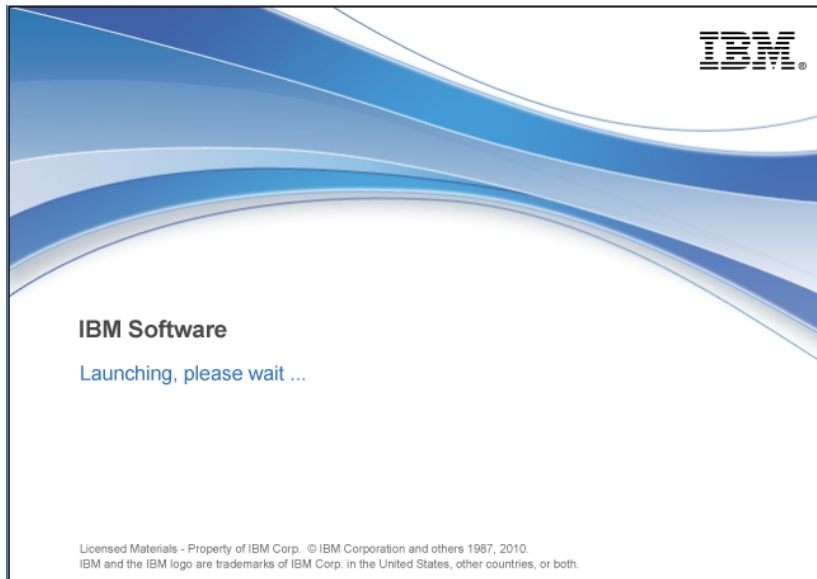
1.2 Getting to know the Toolkit

The icon for the IBM Integration Toolkit is located in the system tray on the desktop. In later labs, you will also be using some of the icons that have been placed on the smart bar.

- ___1. **Click the icon** to start the IBM Integration Toolkit 9.0.

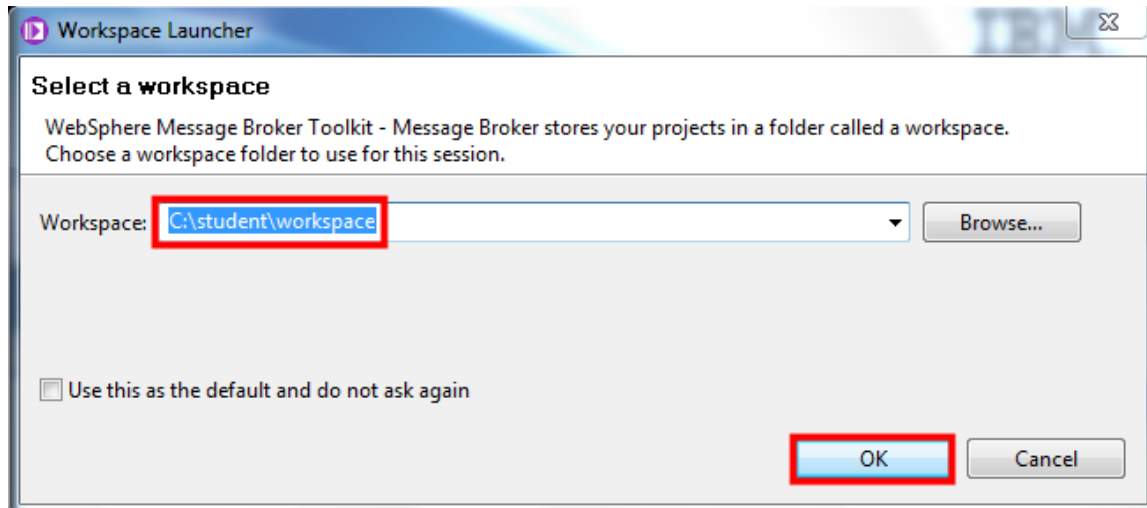


The splash screen is displayed when starting the IBM Integration Toolkit.

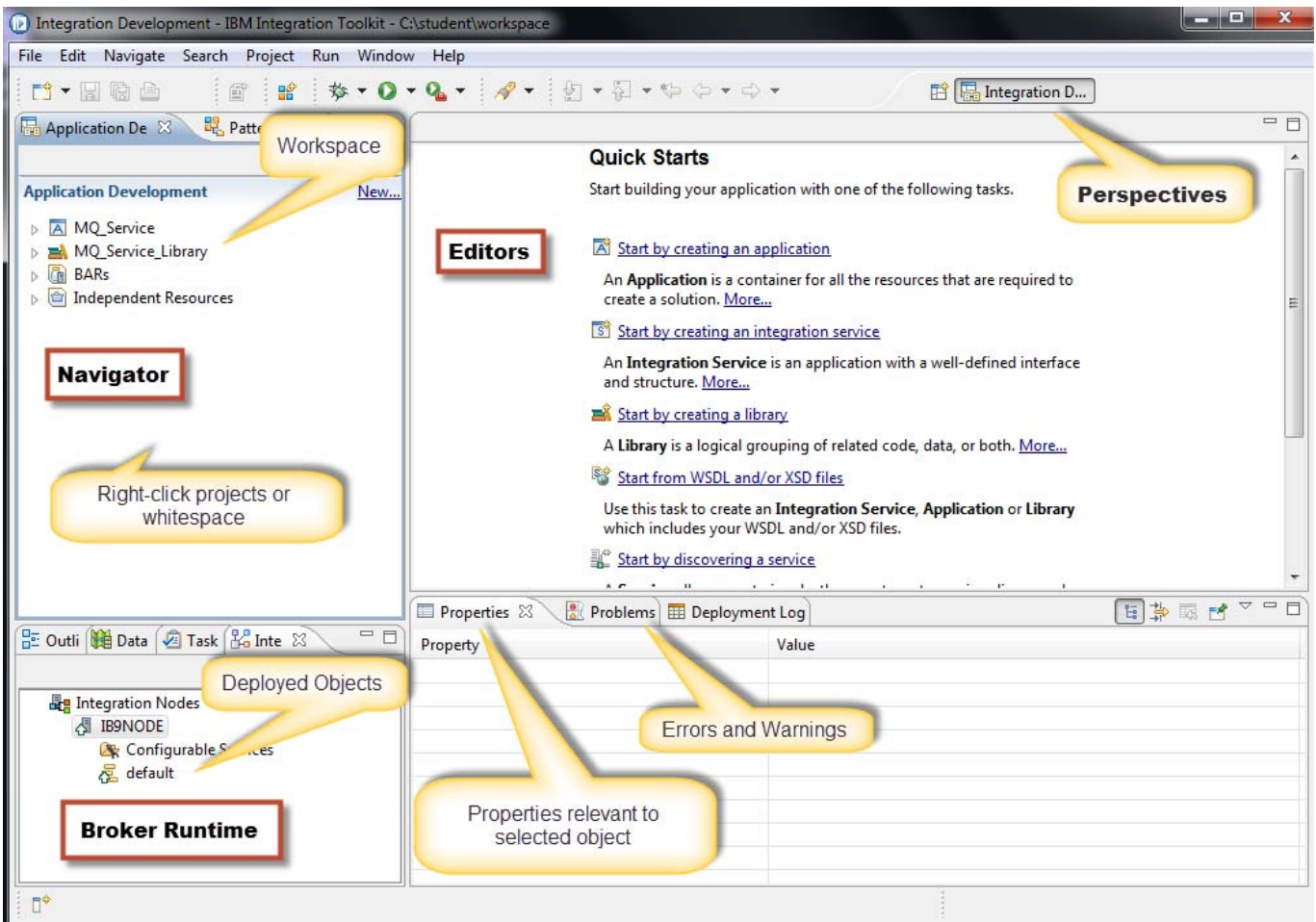


You are prompted to select an Eclipse Workspace – you will take the default here but this is a nice facility to allow you switch between workspaces when starting the toolkit.

- __2. Verify that the **C:\student\workspace** directory is selected.
- __3. Click **OK**.



4. Now take a look around at the Toolkit.



Key Idea: The Toolkit

The Integration Toolkit is based on Eclipse and includes components from IBM Rational® Application Developer. It provides one perspective specifically for IBM Integration Bus as well as additional Perspectives from Rational Application Developer and Eclipse. This system is using the default installation. During the labs and lectures, you will be learning more about the components in a typical development and runtime environment.

The screen capture on the previous page is of the **Integration Development** perspective. It is divided into multiple views (or panes). Each view is identified by a tab at the top of the view. On the lower left is the outline view

On the upper left is the Navigator view, which has tabs for projects (**Integration Development**) and patterns (**Patterns Explorer**). The Navigator view contains the projects that are available within the Eclipse workspace. There is a set of resources already provided for your use during the labs.

The area below the navigator view is the summary area. The **Integration Nodes** tab will show all defined local nodes as well as connections to remote nodes that have been created.

The large area on the right is used by the resource editors. When an editor has opened a resource, it will also be represented by a tab. Below the editor view is a pair of views for Properties and Problems.

On the top right are tabs for the perspectives that have been opened. To change an open perspective, you can simply click its tab.

Quick Start wizards are displayed as links in the center of the screen, which is the default blank palette within the Editor view. You can also access these by right clicking whitespace in the Applications view on the left. The wizards do some of the initial work for creating various types of solutions.

The Integration Toolkit provides several **Quick Start** wizards plus a number of pre-defined patterns to assist in creating Applications and Libraries. We will see more of these in later labs.

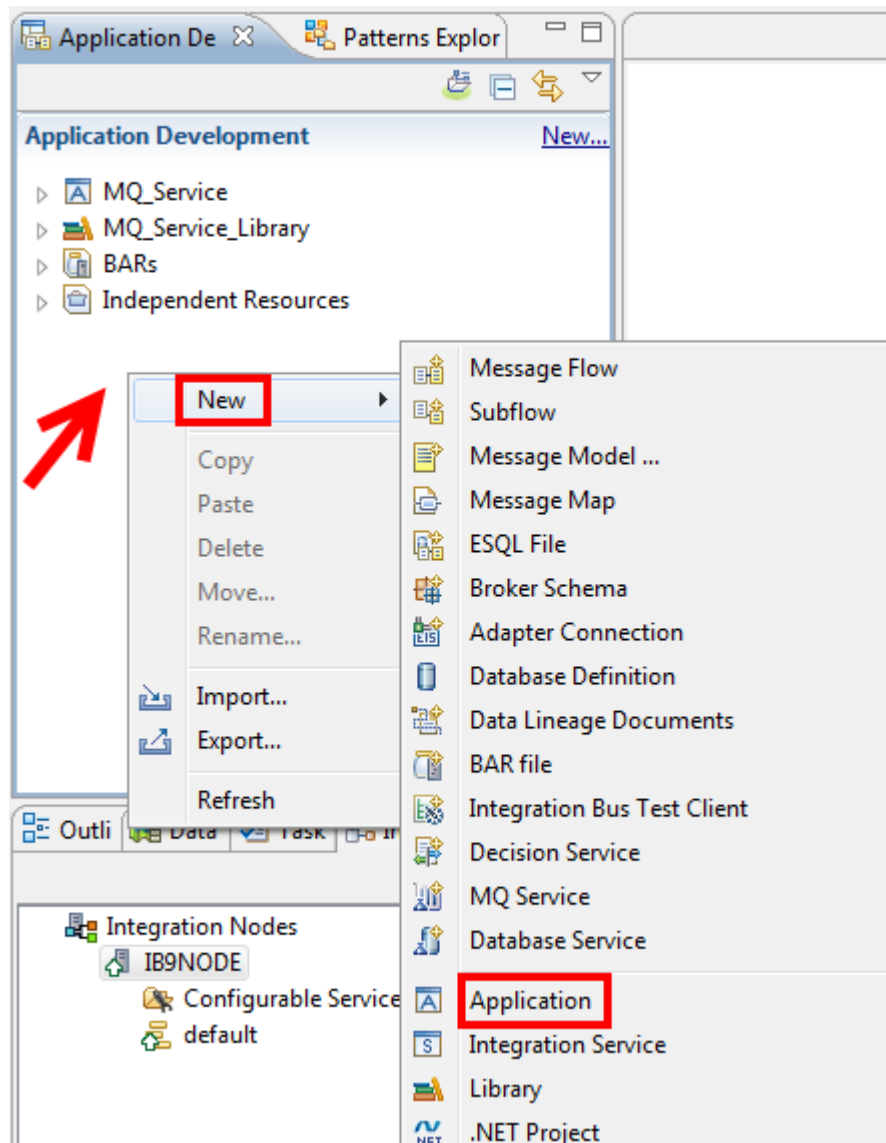
Eclipse is project-oriented – artifacts are organized into projects. A project is typed. Project types that are specific to IBM Integration Bus are Applications and Libraries. Also, legacy projects of type Message Flow Project and Message Set Project can be created or imported into the toolkit. Since they pre-date the concept of Applications and Libraries, they will be visible in the hierarchy under a Folder called “Independent Resources.”

1.3 Building a simple flow

- ___1. Right click the white space of the Application Development pane.
- ___2. Select **New → Application**.

As an alternative, you can click **File** on the Menu Bar, and select **New→Other**, type **Application** in the selection box, and click **Next**.

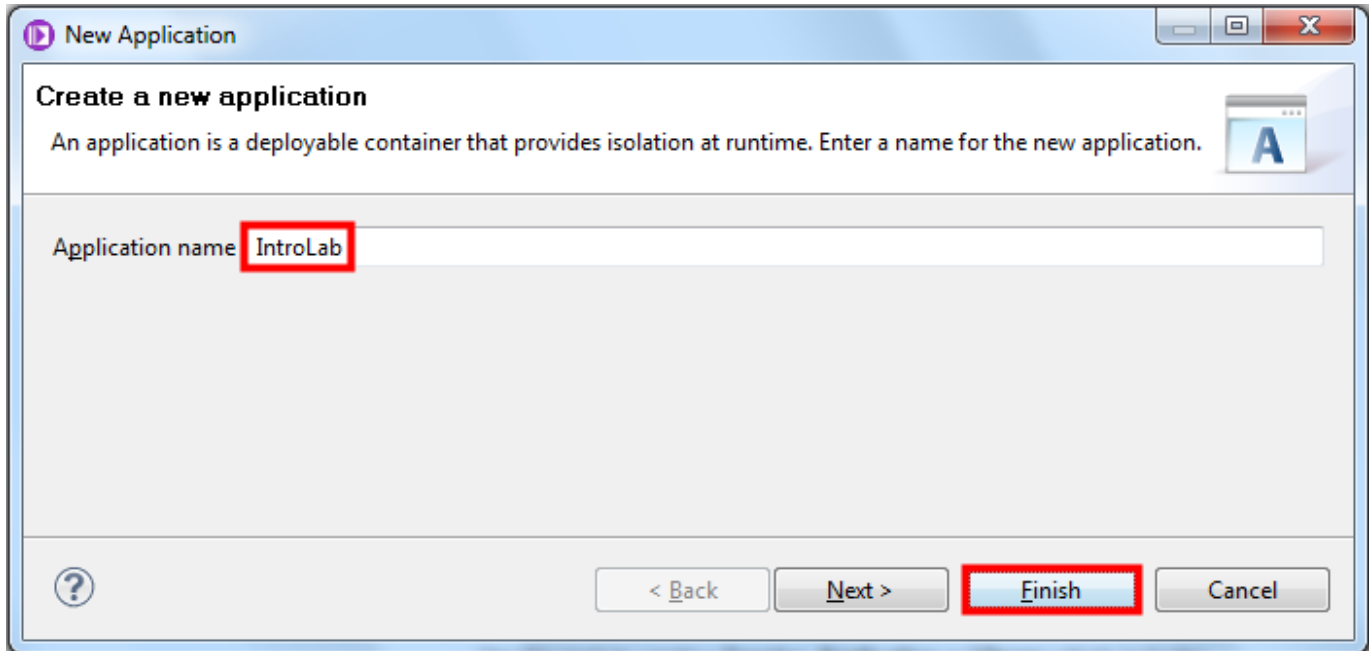
Note: These actions are also available as icons on the toolbar.



You are prompted to enter a name for your application.

__3. Enter **IntroLab** for the application name.

__4. Click **Finish**.

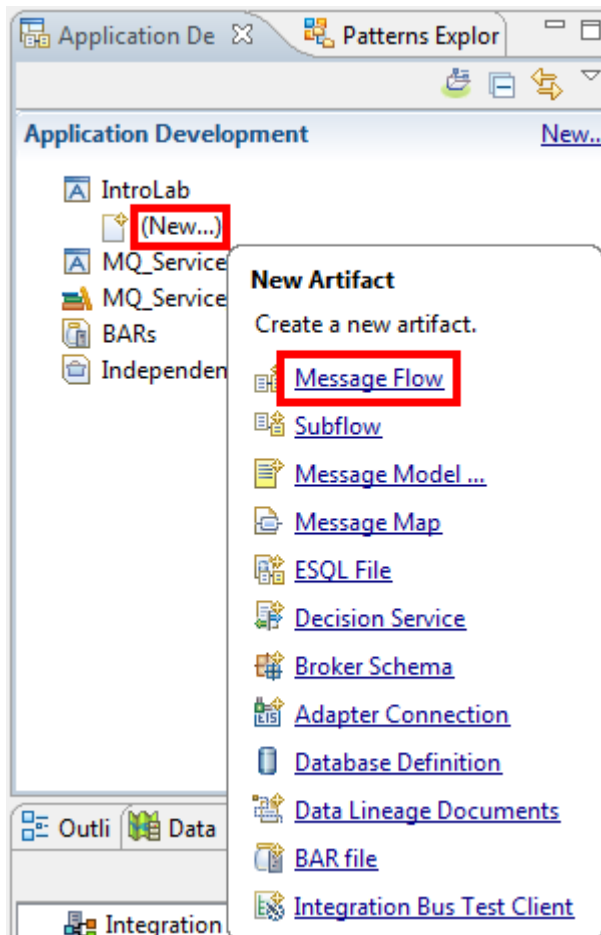


Now you will create a new message flow.

__5. Under the **IntroLab**, click **(New...)**.

__6. Select **Message Flow**.

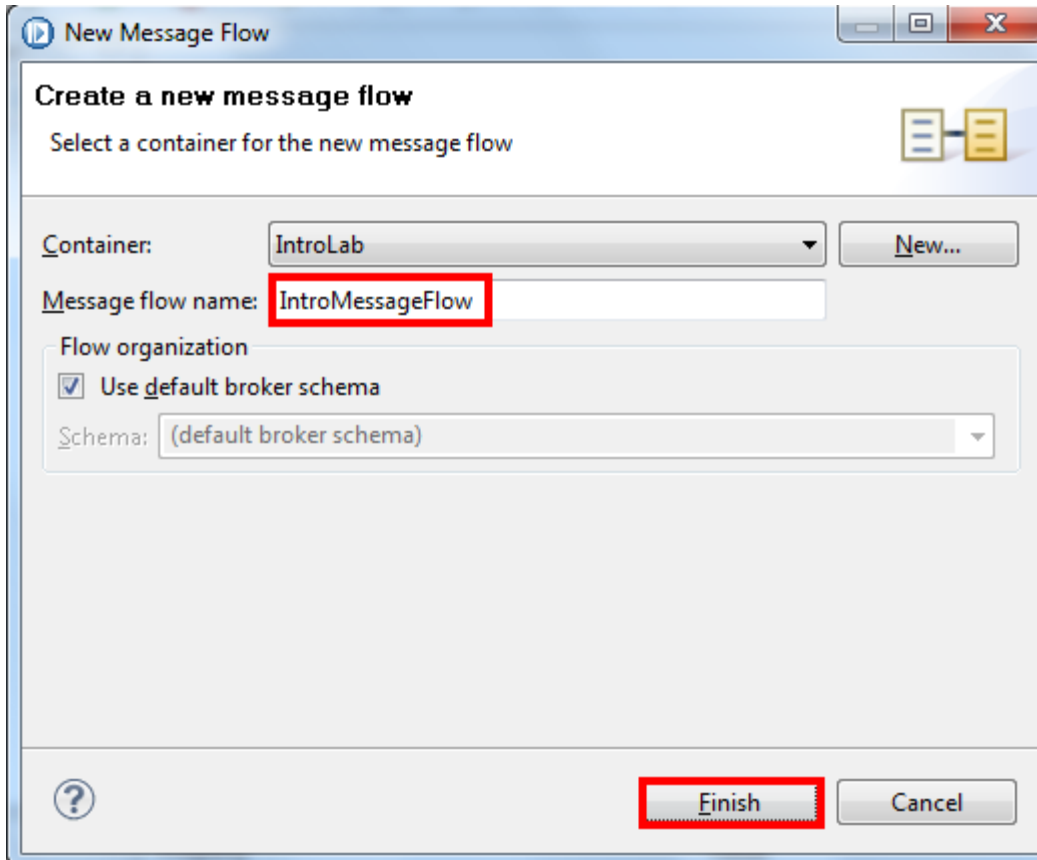
The options for the New action differ depending on how the request is made. For example, when using the File → New from the Menu bar, all of the options will be listed. However, in this case, by starting from an application, the only options shown are those that are related to the selected project.



Here you are asked to name the message flow. An **Application** might contain multiple message flows.

__7. Enter **IntroMessageFlow** in the Message flow name box.

__8. Click **Finish**.



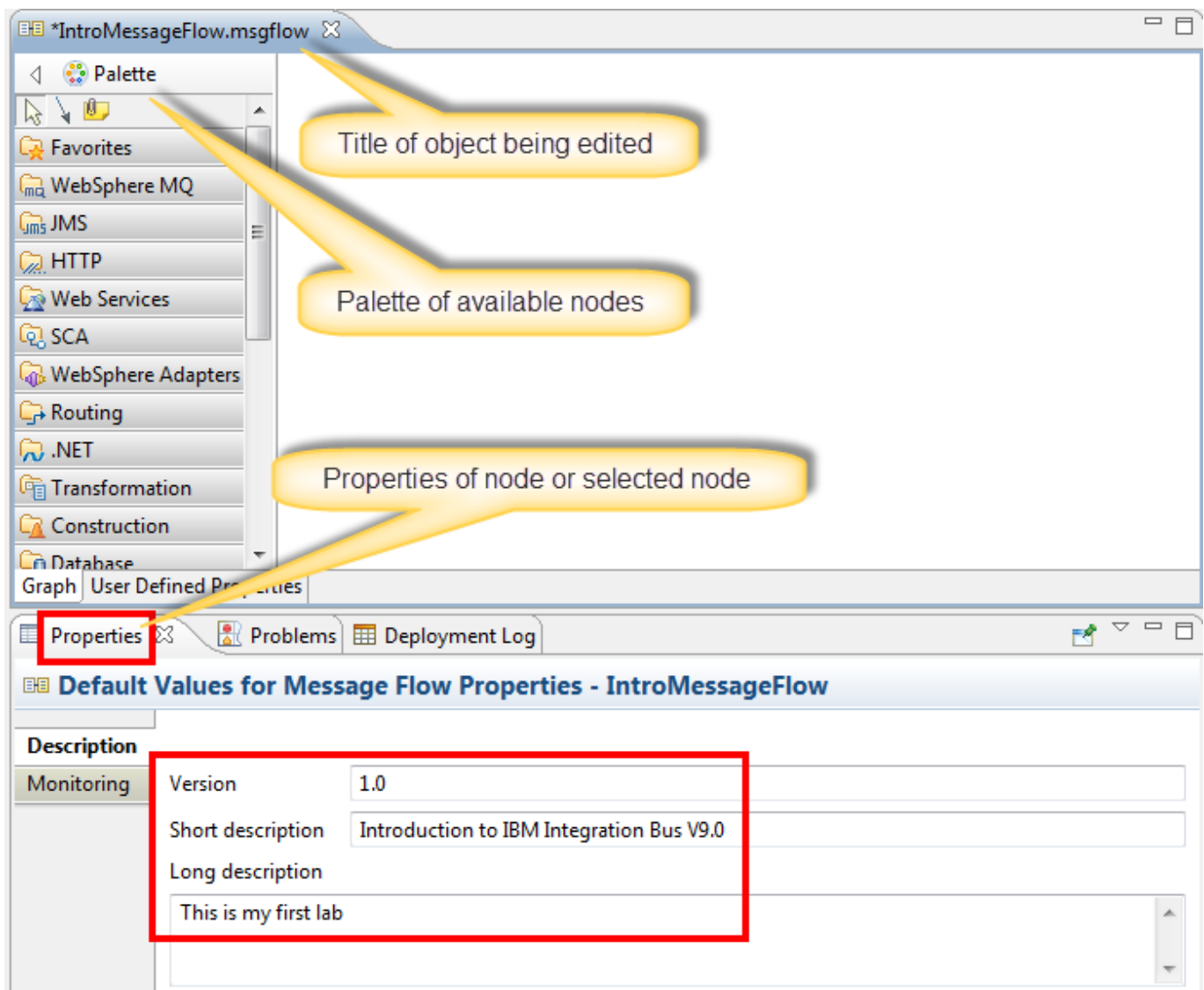
You are placed into the Message Flow Editor where you can compose the message flow. When you click the Message Flow Editor, information about the message flow appears in the Properties pane.

__9. Select the **Properties** tab.

__10. Select the **Description** tab.

__11. Enter the following:

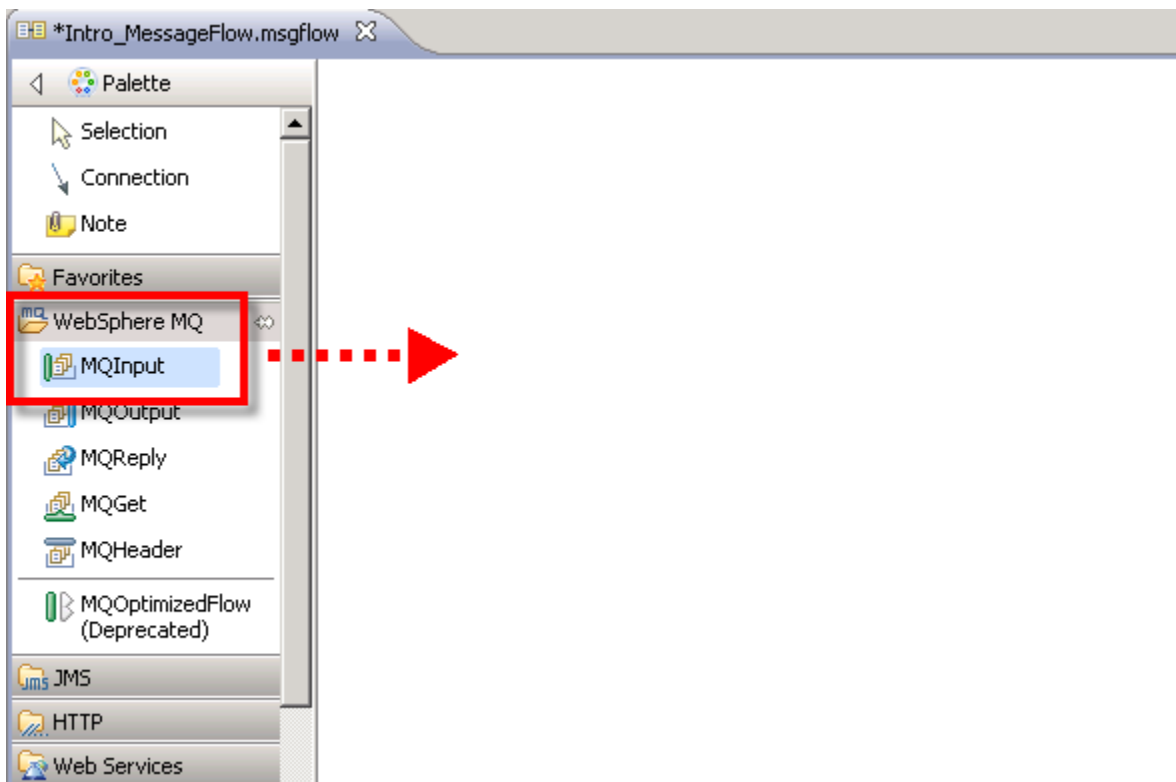
- Enter **1.0** for the **Version** field;
- Enter **Introduction to IBM Integration Bus V9.0** for the **Short description** field;
- Enter your choice of information in the **Long description** field.



A message flow must begin with an Input node. An input node establishes the environment for the flow. There is an Input node for each of the various protocols that IBM Integration Bus supports. We will process a WebSphere MQ message with this flow so we need an MQInput node.

The MQInput node is in the WebSphere MQ drawer.

- __12. Open the **WebSphere MQ** drawer by clicking it. If a drawer is open, it will close when clicked.
- __13. Highlight the desired node (**MQInput**).
- __14. Either drag it to the canvas or move the mouse to the canvas and click again.



When a node is initially added, its name can be changed immediately by over-keying the default name – or by entering a new value in the Node name field in the Description tab of the Properties.

A best practice is to provide a new name for each node that is descriptive of the function that it provides. For most of the labs, you will be renaming the nodes. If you use the names as suggested it will make it easier to follow the lab guide. Another naming convention for MQInput and MQOutput nodes is to use the queue name that the node is accessing.

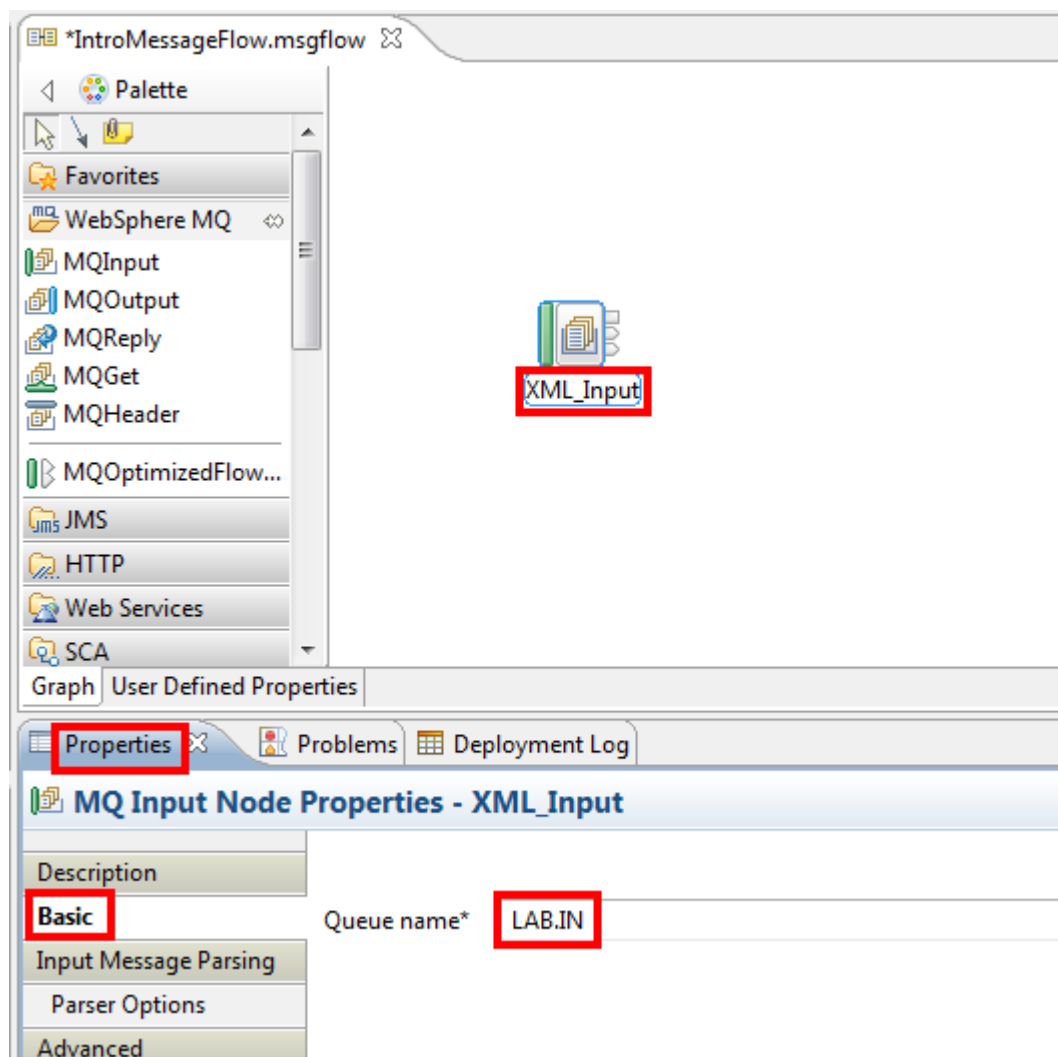
__15. Change the name of the node to **XML_Input**.

__16. Press the **Enter** key to complete the rename.

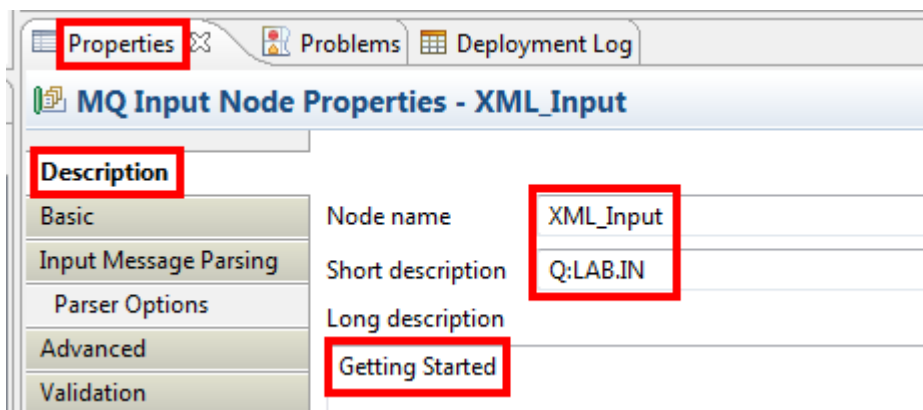
The **Queue name** in the node properties must be set. The **Basic** tab should be selected. A Queue name is required and this is indicated by a message in red.

__17. Select the **Basic** tab in the **Properties** pane.

__18. Enter **LAB.IN** as the **Queue name**. Queue names are case sensitive. All queue names in the labs are upper case.

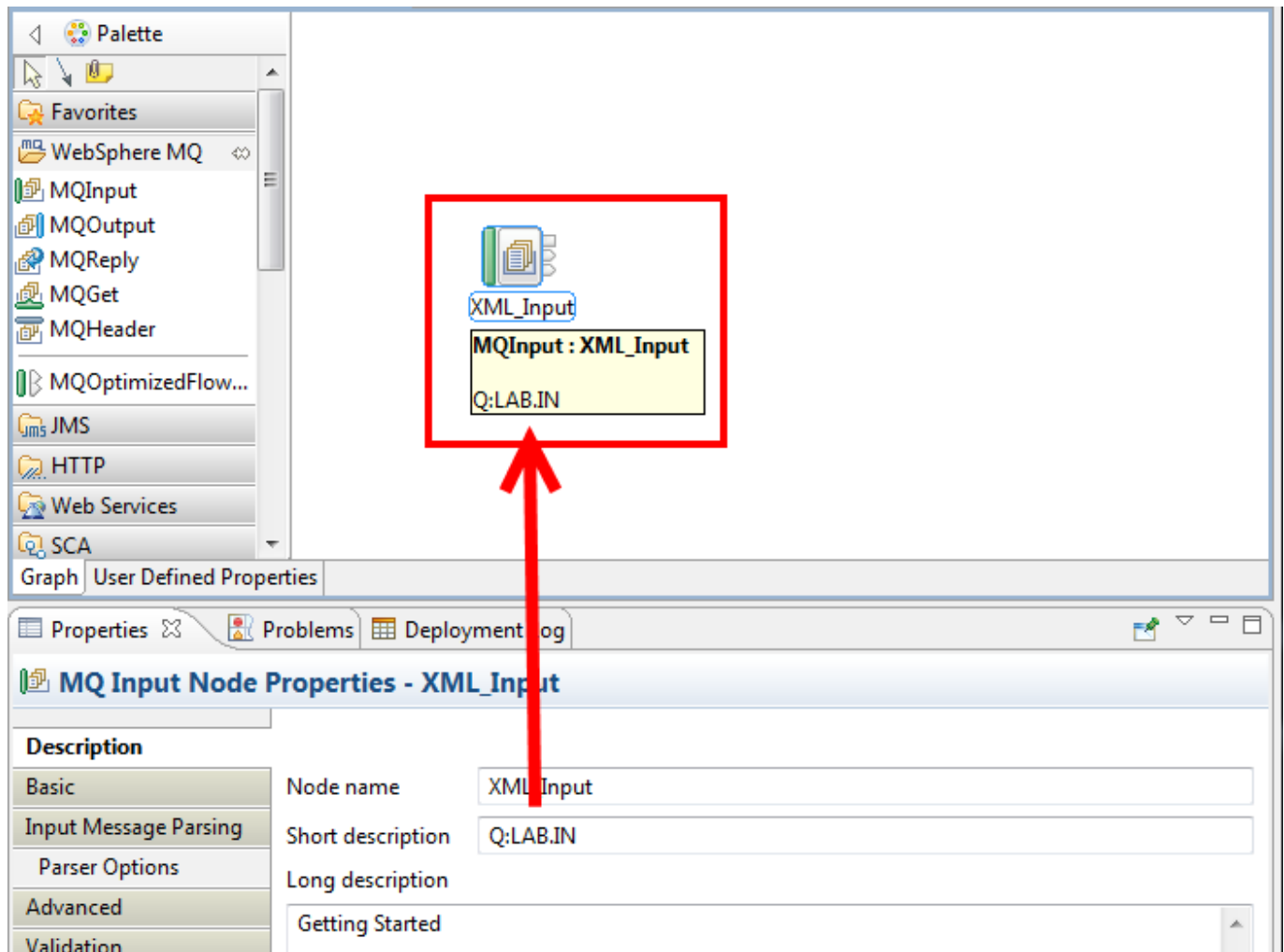


- ___19. Select the **Description** tab. This section is used for documentation. The name of the node could also be changed. The node name is shown in the message flow editor. It does not affect the operation of the message flow.
- ___20. Enter **Q:LAB.IN** in the **Short description** field.
- ___21. Enter your choice of text for the **Long description** (**Getting Started** is shown in the screen shot).



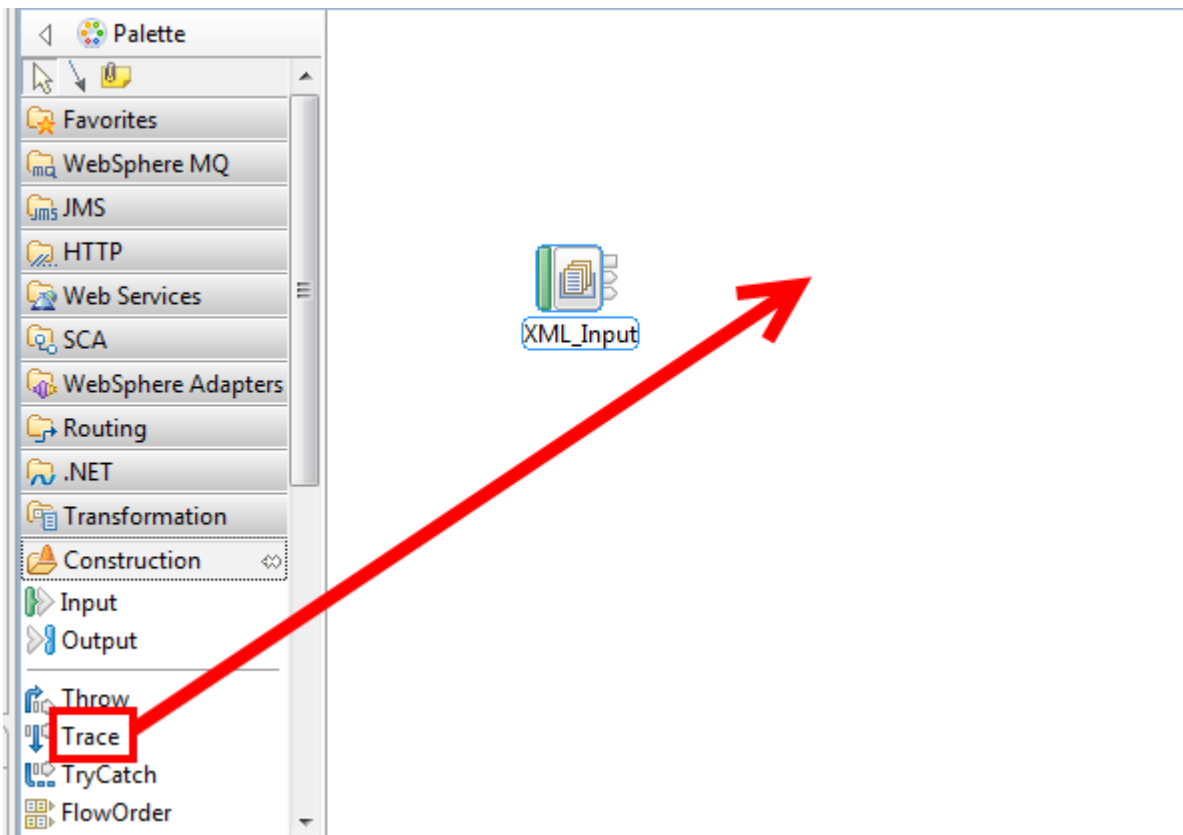
22. Hold the mouse pointer over the node name.

The information in the Short description field is displayed. When there are multiple nodes on the canvas, if you move from node to node with the mouse, the same tab in the Properties will be displayed.



The **Trace** node is in the **Construction** drawer.

- __23. Open the **Construction** drawer by clicking it with the mouse.
- __24. Select the **Trace** node and place it on the canvas to the right of the **XML_Input** node. As shown in the example, when you place the cursor on a node name, a description is shown. You do not need to rename the Trace node.
- __25. Press the **Enter** key to accept the default node name (**Trace**).



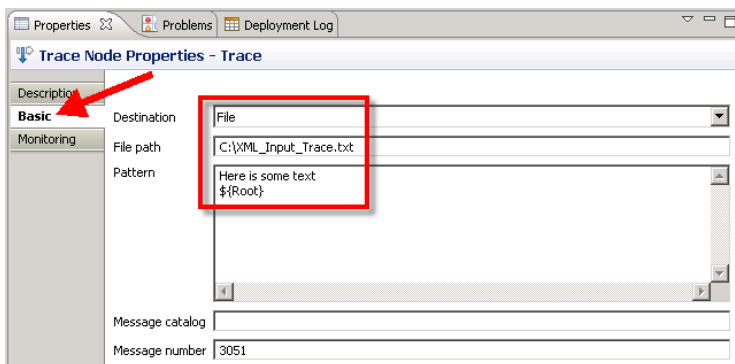
__26. In the **Basic** tab, use the pull-down list on the **Destination** field to select **File**.

__27. In the File Path field, enter **C:\XML_Input_Trace.txt**.

The information in the **Pattern** box tells the node what information to produce in the trace output. If you type a line of raw text, it is echoed to the output.

__28. Enter a line of your choice in the **Pattern** box – no quotes are needed.

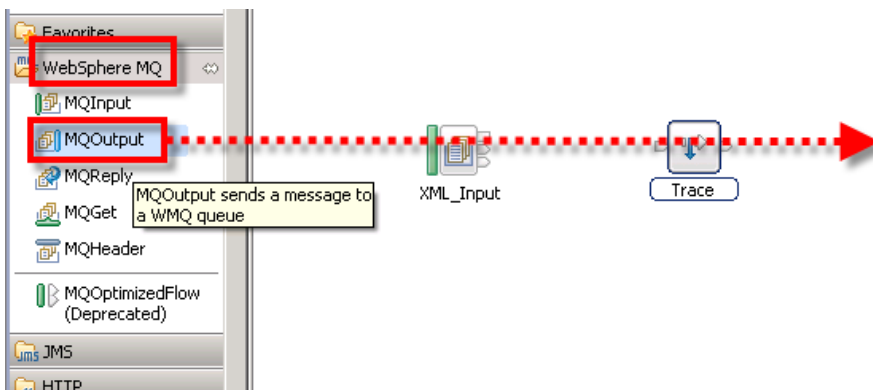
__29. In the **Pattern** box, enter the string **\${Root}** exactly as indicated – this tells the trace node to dump out the entire contents of the message that enters the node. Important – the pattern uses **curly braces**, not parenthesis. The expression between the curly braces will be evaluated at run time.



__30. Open the **WebSphere MQ** drawer.

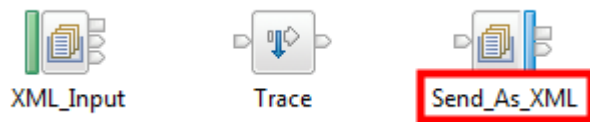
__31. Select an **MQOutput** node from the drawer.

__32. Place it to the right of the **Trace** node.



__33. While the node name is highlighted, enter **Send_As_XML** as the new name.

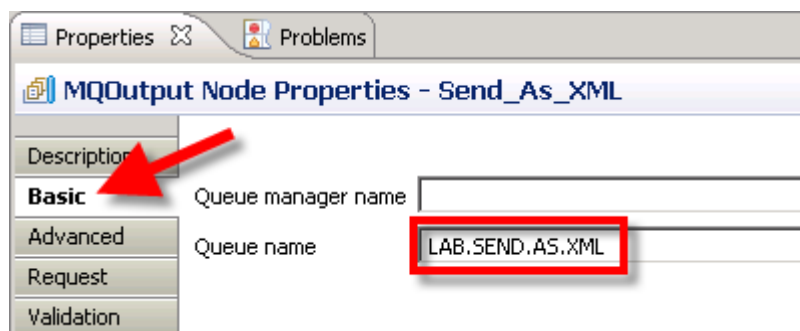
__34. Press the **Enter** key to accept the new node name.



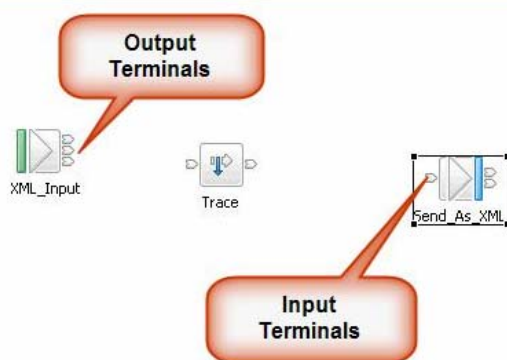
__35. Click the **Basic** tab on the **Properties** tab.

__36. Set the **Queue name** to **LAB.SEND.AS.XML**. Queue names are case sensitive. It is a best practice to separate words in the queue name with a dot.

__37. Make sure you did not enter this information in the **Queue manager name** field,



Key Concept: Node Terminals



As you work with the various nodes, you will also be working with their Input and Output terminals. Input terminals are typically named **In**. Most nodes have an Output terminal named **Out**. They may have several others.

Some of these will have common names such as **Failure** or **Catch** and others will be unique to that particular node. Some nodes allow you to define the terminals. The terminals are given a name when they are defined.

The lab instructions will identify the Output terminal to be used when connecting nodes together. If you hover the mouse pointer over a terminal, a small popup will appear that identifies the name of the terminal.

You will now wire the nodes together to create a logical path for the message to follow through the message flow. The **Out** terminal of the XML_Input node must be wired to the **In** terminal of the Trace node. There are two techniques:

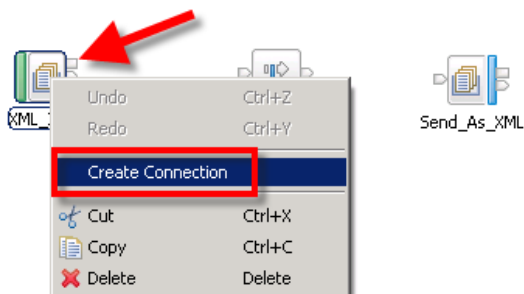
One – position the mouse over the Out terminal (in the middle), click and drag to the target and click again.

Two – right click a node and select **Create Connection**. This is an example of a Terminal Selection presented as a result of the Create Connection.

The rest of the Lab instructions show the first method for wiring.

__38. Right click the **XML_Input** node.

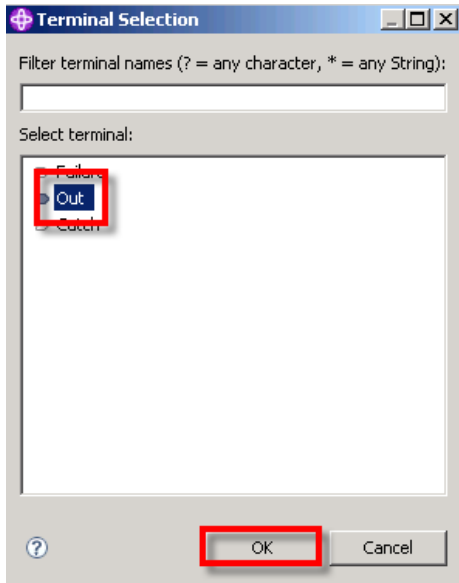
__39. Select **Create Connection** from the menu.



A list of the available Output terminals for this particular node type is shown

__40. Select the **Out** terminal.

__41. Click **OK**.

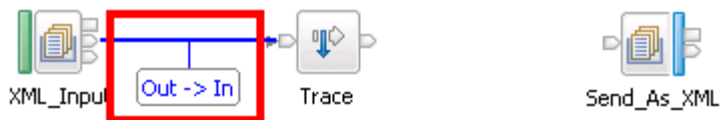


You now have a “rubber-band” connector.

__42. Position the connector over the **Trace** node.

__43. Click the left mouse button to anchor it, creating a connection between the two nodes.

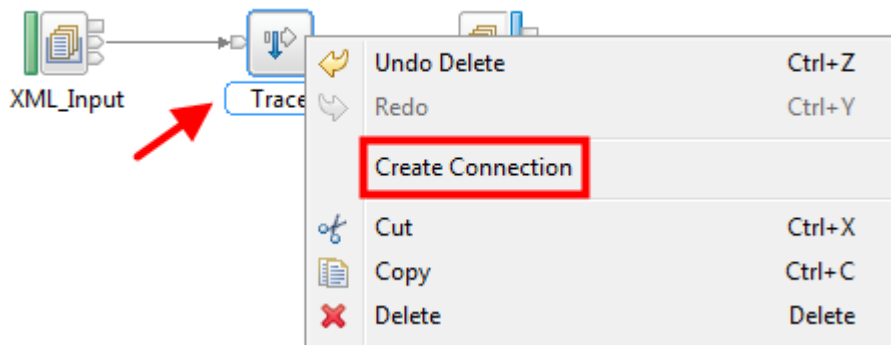
__44. Place your mouse pointer on the connection. A popup of a summary of “from-to” information will appear.



The same steps will be used to make a connection from the **Trace** node to the **Send_As_XML** node.

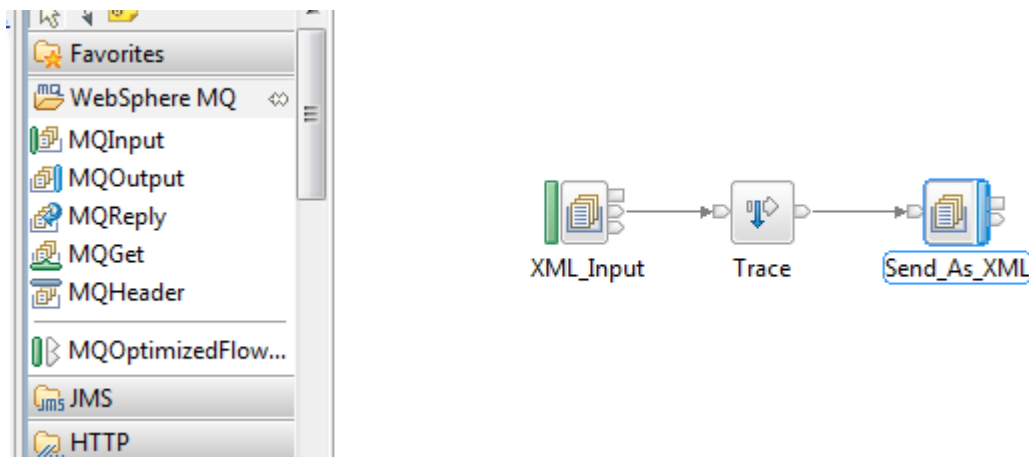
__45. Right click the **Trace** node.

__46. Select **Create Connection**. This time you immediately get a “rubber-band” connector. No selection list of terminals is presented because there is only an **Out** terminal on the Trace node.

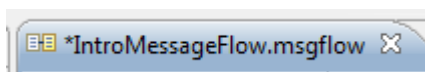


__47. Position the mouse pointer over the **Send_As_XML** node.


__48. Click the left mouse button to create the connection.



Your message flow should now look like the above diagram. Notice the small asterisk next to the message flow name. This indicates that the message flow has not been saved to disk.



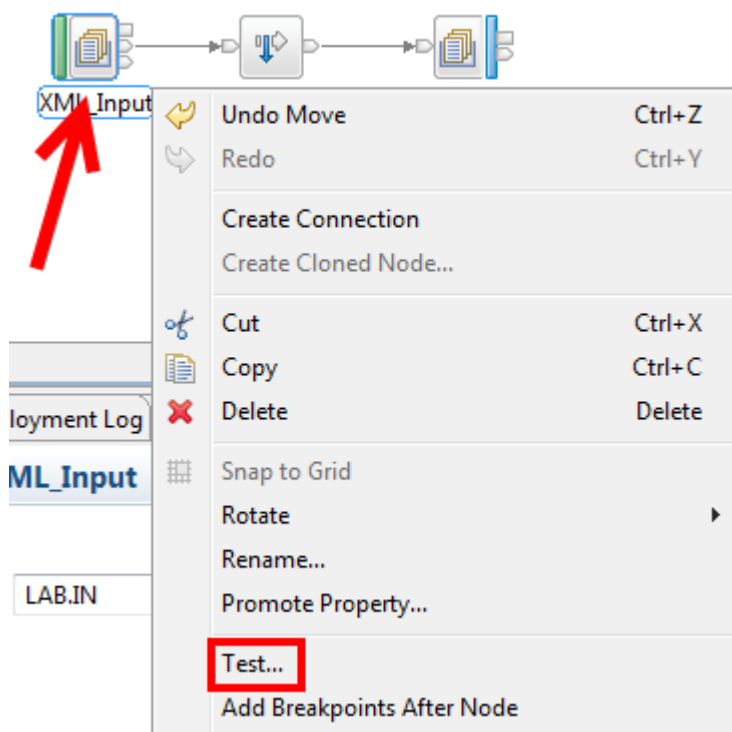
__49. It is time to save your work – hold down the **Ctrl** key and press the **S** key to save the message flow. You can also click the “diskette” icon or use **File** → **Save** to perform a save.

The following graphic will be used as a reminder when it is time to save your work. 

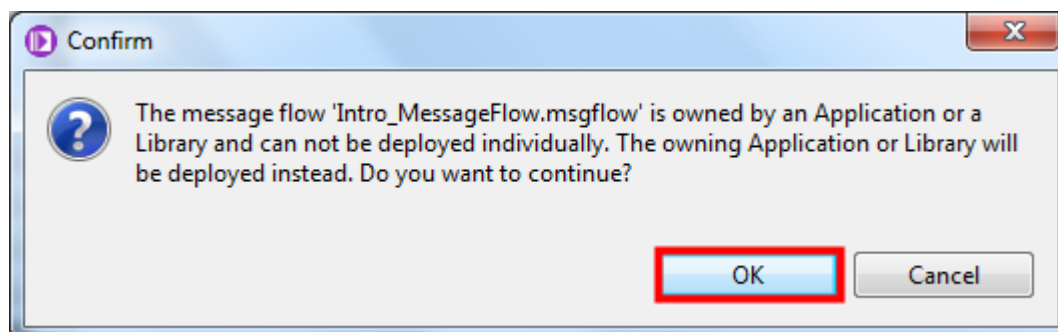
1.4 Testing the flow

The message flow is now complete. The next step is to send it to the runtime environment for testing. The integrated Test Client will be used to test the message flow.

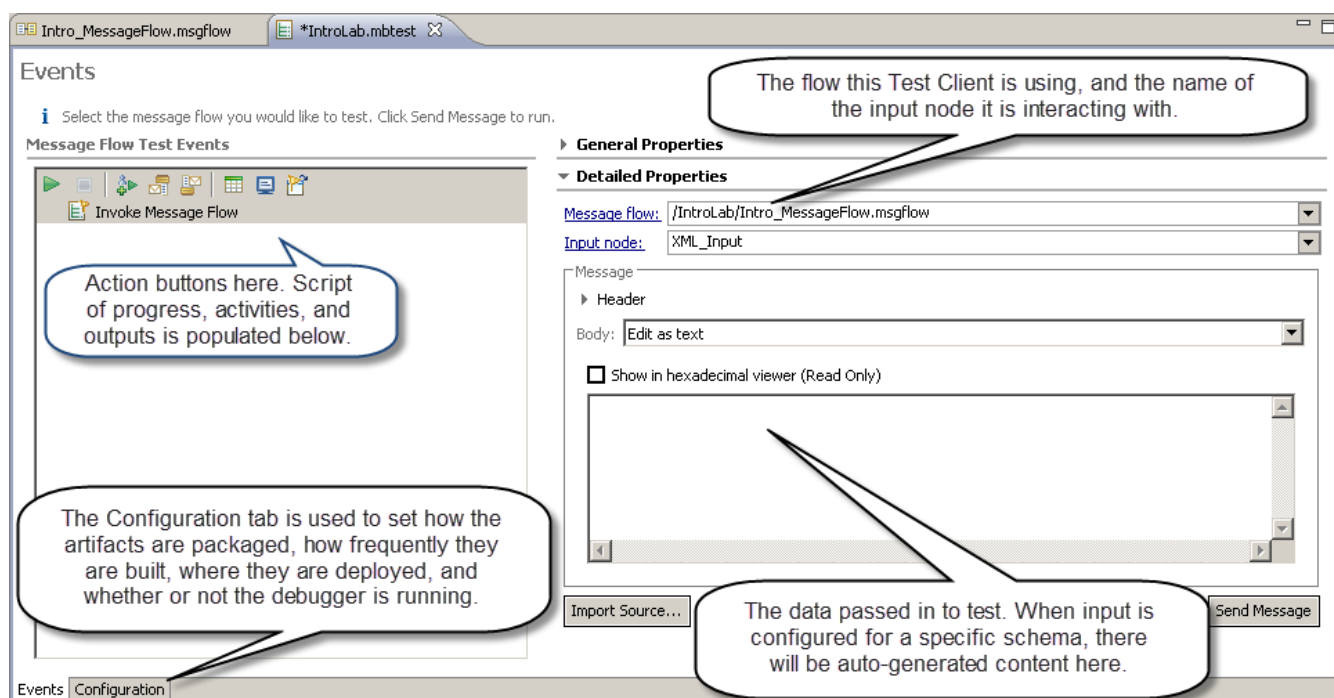
- __1. Select the **MQInput** node (which we called **XML_Input**).
- __2. Press the right mouse button.
- __3. Select **Test...** from the menu.



- __4. Click **OK** to the popup.



Take a moment to familiarize yourself with the Test Client:

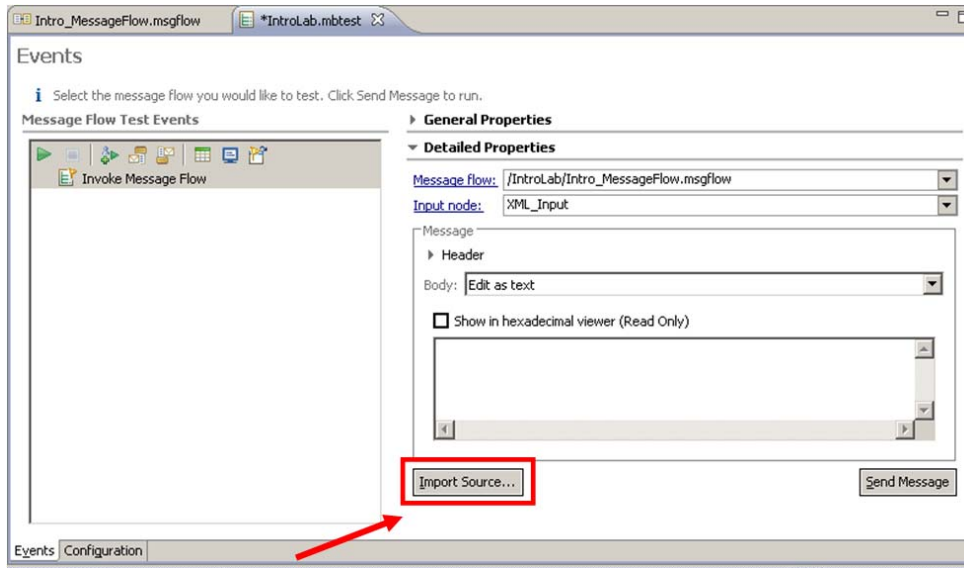


Key Concept: Test Client

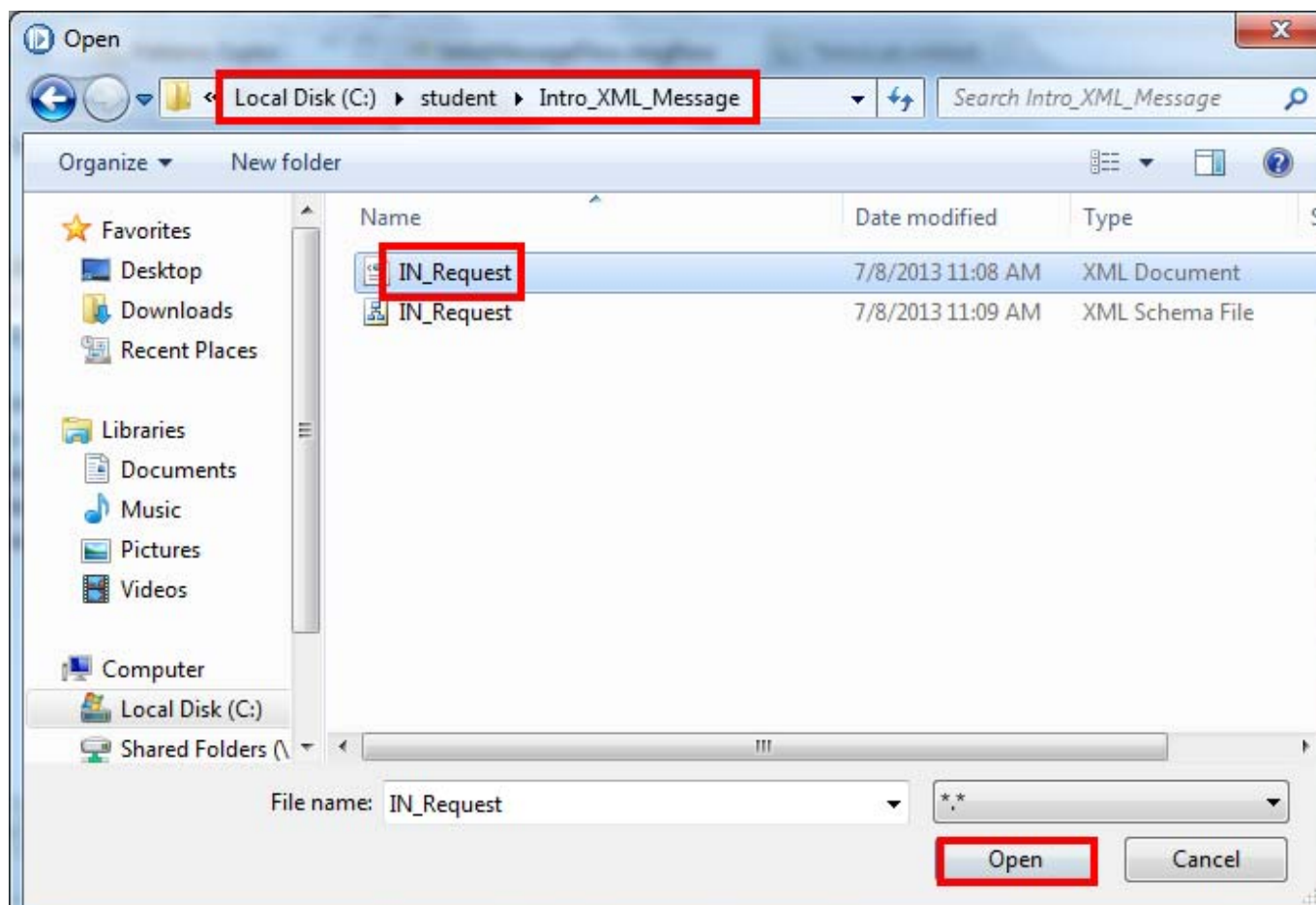
Test Client instances can be created for MQ, JMS, HTTP, SOAP and SCA input nodes. They exist as a single file in the workspace with a .mbtest file extension. They can be embedded into the Applications or Libraries and thus passed from developer to developer with a project.

The Test Client has many useful features such as monitoring all supported output paths through a flow and storing sample messages to a data pool. It also encapsulates the build and deploy process, and can be used to launch the debugger. Since it encapsulates a full test, including data, it can be used for regression testing.

5. Select Import Source...

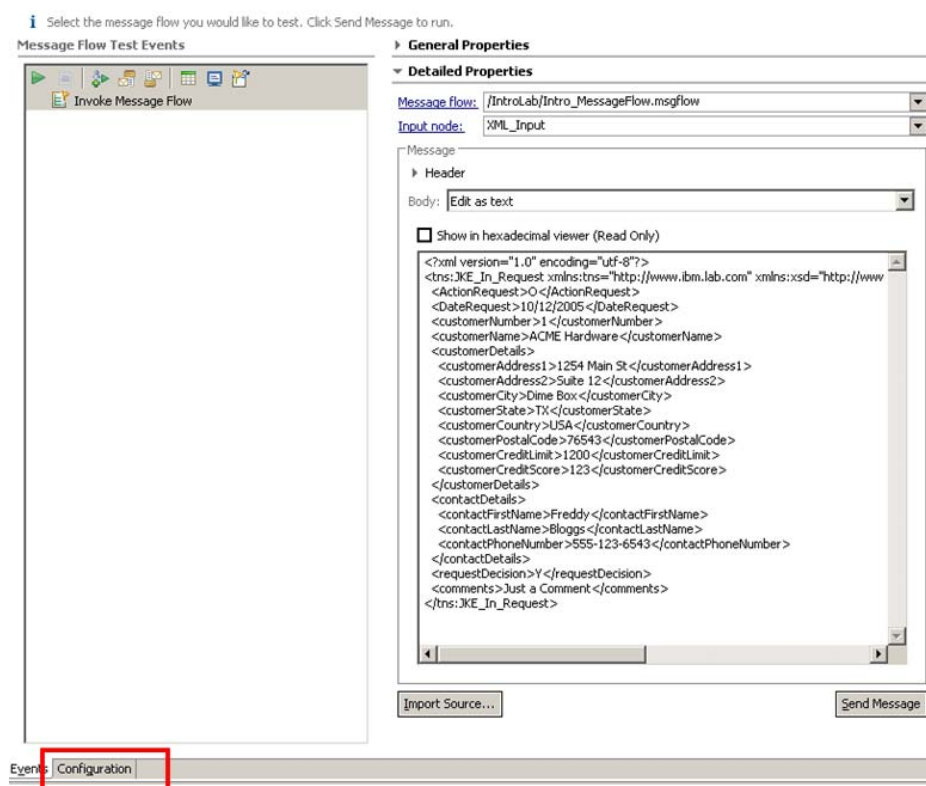


- __6. Navigate to the **C:\student\Intro_XML_Message** folder.
- __7. Select **IN_Request** file. The **Type** should be **XML Document**.
- __8. Click **Open**.



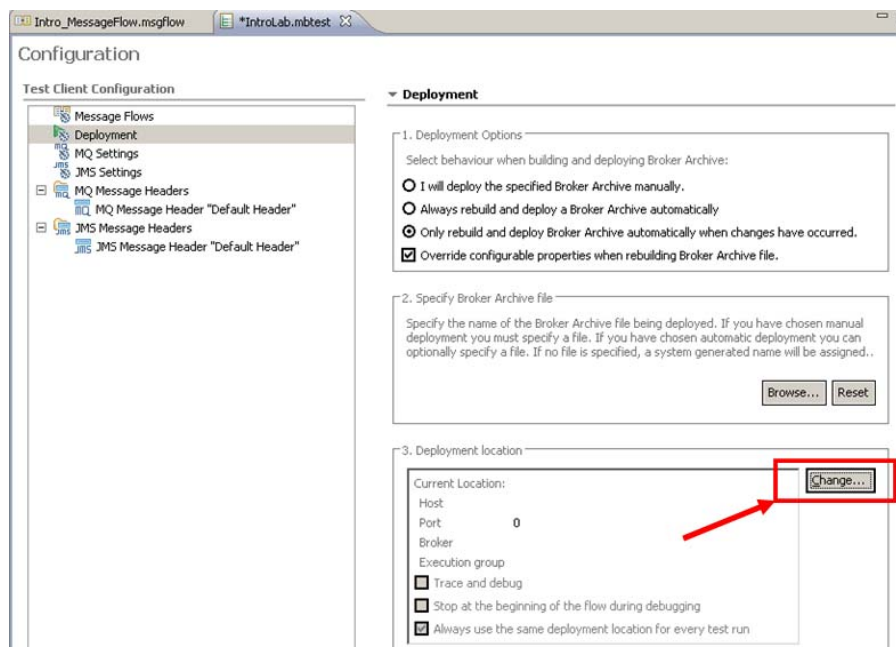
The XML sample message is now loaded into the Test Client.

__9. Select the **Configuration** tab.



__10. Select **Deployment**.

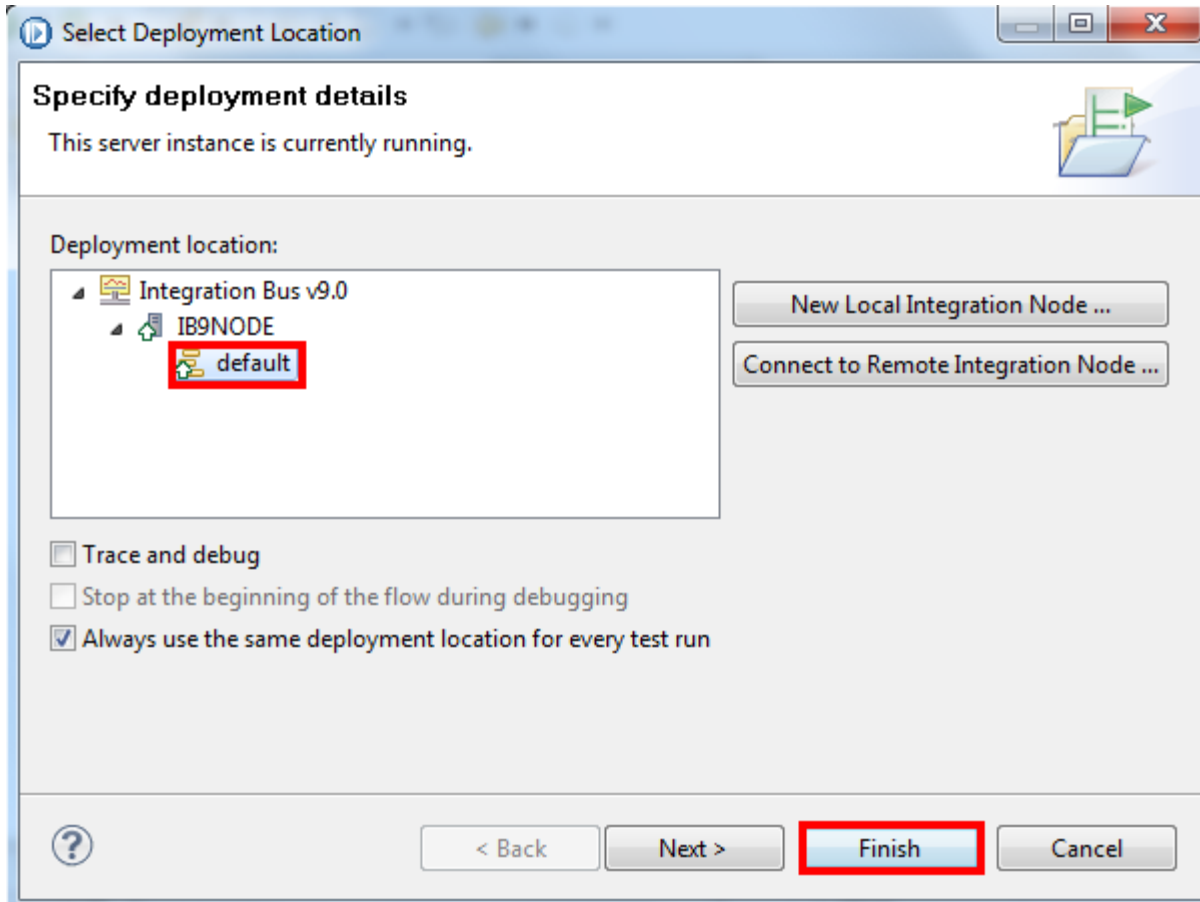
__11. Click the **Change** button in **Section 3: Deployment Location**.



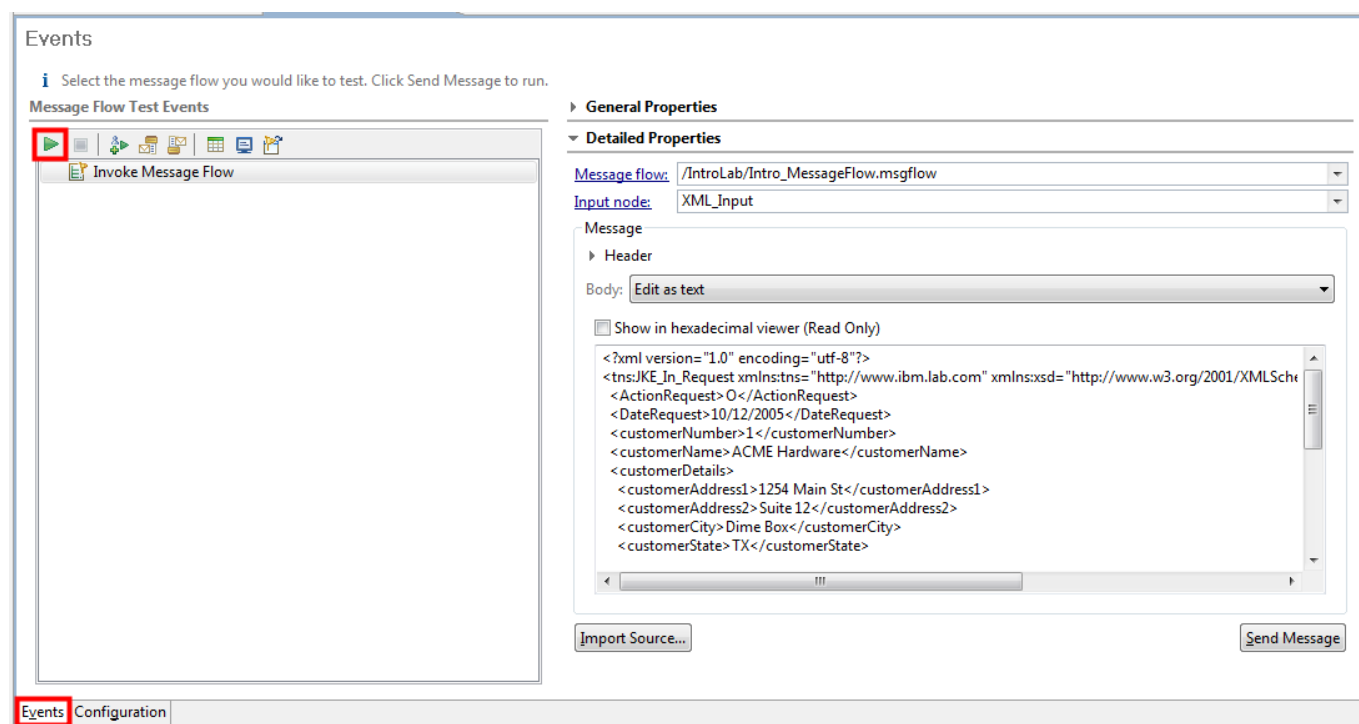
Note that the local Integration Node (**IB9NODE**) and the Integration Server (**default**) are displayed.

__12. Select the **default** integration server.

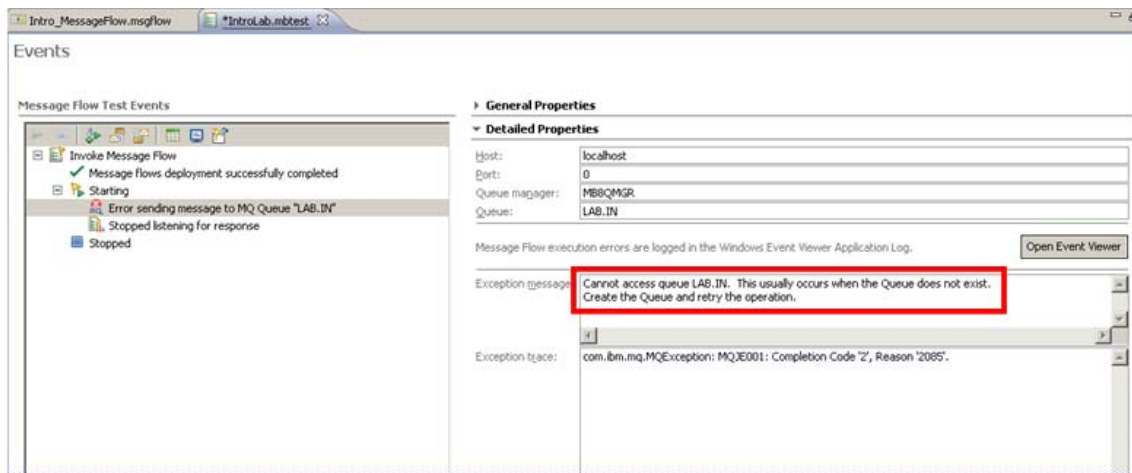
__13. Click **Finish**.



- ___14. Switch back to the **Events** tab.
- ___15. Click the green **Play** button. (The **Send Message** button will also do the same thing.)

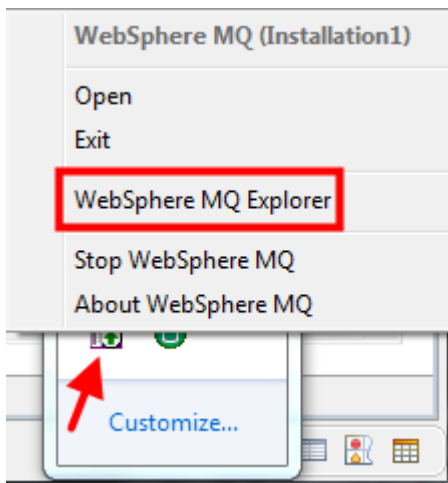


What happened? We received an error when the Test Client attempted to launch. It could not connect to the queue and gives us a clue by asking whether or not it has been created. We did indeed forget to create our queues!

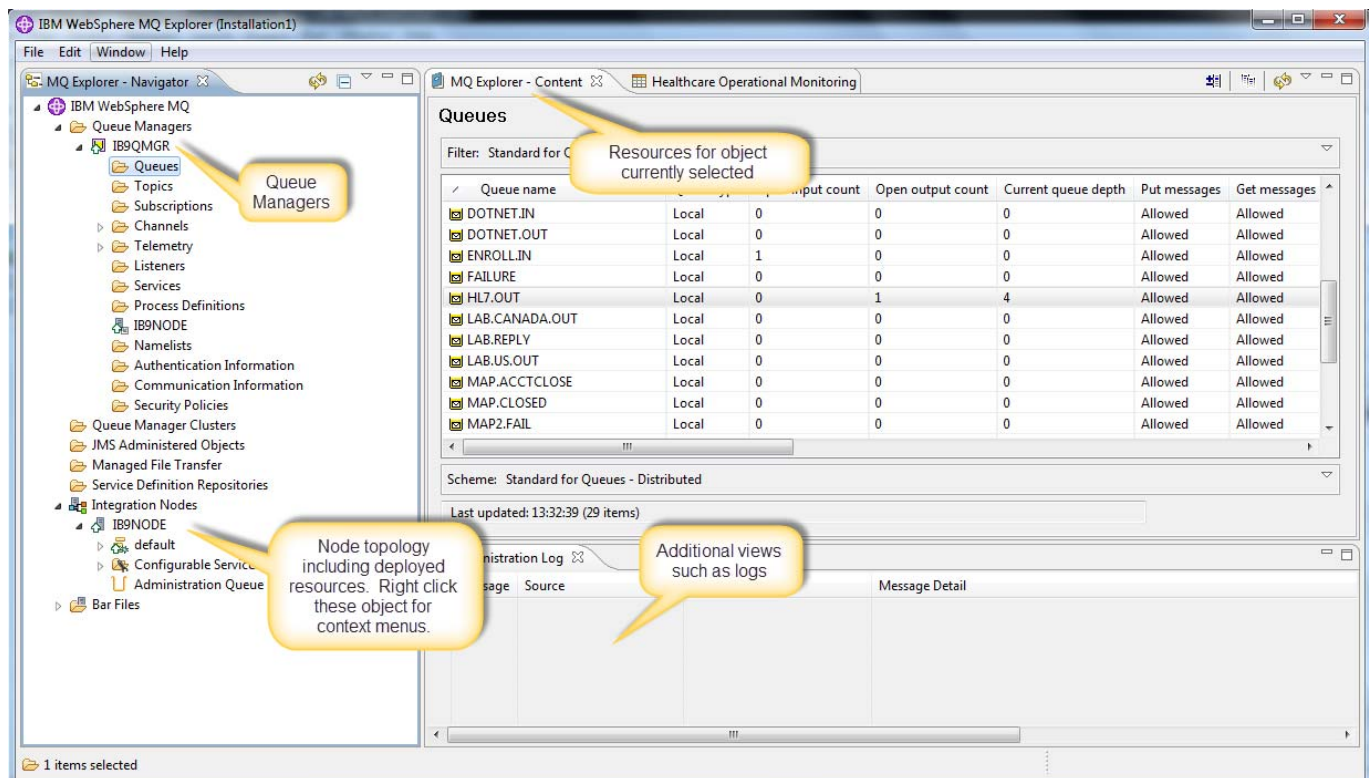


To create the queues, we will switch to the Administrator tool called the IBM WebSphere MQ Explorer, which includes Integration Bus Explorer as well.

- __16. Find the WebSphere MQ Alert Monitor in the Windows system tray.
- __17. Press the right mouse button.
- __18. Select **WebSphere MQ Explorer** from the menu.



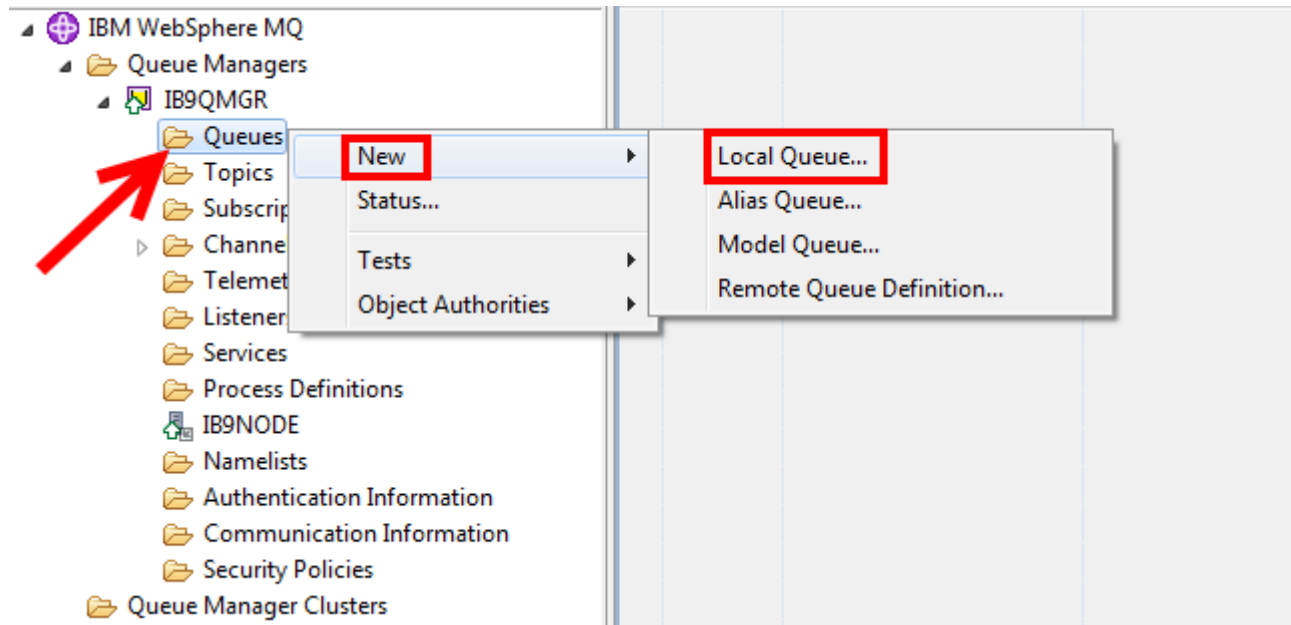
Take a moment to familiarize yourself with the Explorer. This tool will be used again in later labs.



Key Concept: Integration Explorer (sometimes also called MQ Explorer)

The Integration Explorer is a lightweight Eclipse tool for the administration of your Integration Bus and MQ topology. Technically, the Integration Explorer is an extension to the MQ Explorer that adds Integration Bus administrative capabilities to the tool. A few examples of these tasks include adding, modifying or deleting Brokers, access control, the deployment and modification of Apps and Libraries, viewing Administrative and Event logs, and administering the MQ Pub or Sub engine.

- __19. Expand **IB9QMGR** (if necessary).
- __20. Right click the **IB9QMGR→Queues** folder.
- __21. Select **New→Local Queue**.



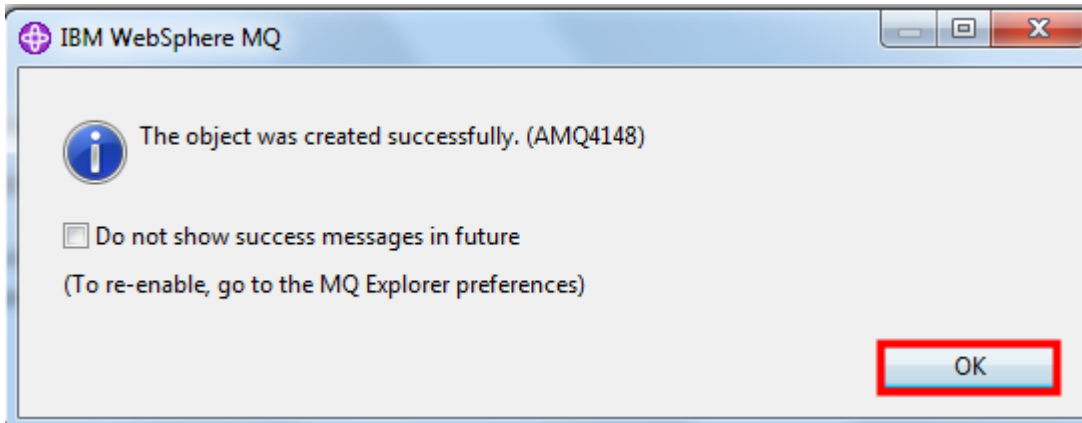
__22. In the dialog, for the **Name** of the queue, enter **LAB.IN**.

__23. Click **Finish**.

The screenshot shows a 'New Local Queue' dialog box with the following elements:

- Title Bar:** 'New Local Queue' with standard window controls.
- Section Header:** 'Create a Local Queue'.
- Instruction:** 'Enter the details of the object you wish to create'.
- Name Field:** A text input field labeled 'Name:' containing 'LAB.IN', which is highlighted with a red rectangular box.
- Select Existing Object:** A section titled 'Select an existing object from which to copy the attributes for the new object.' containing a text field with 'SYSTEM.DEFAULT.LOCAL.QUEUE' and a 'Select...' button.
- Automatic JMS Queue:** A checkbox labeled 'Start wizard to create a matching JMS Queue' which is currently unchecked.
- Navigation Buttons:** A row of buttons at the bottom: '< Back', 'Next >', 'Finish' (highlighted with a red rectangular box), and 'Cancel'. A help icon (?) is located to the left of the 'Back' button.

__24. Click **OK** to close the dialog. The dialog confirms that the queue was created successfully.



__25. **IMPORTANT!** Repeat steps 20 through 24 to create a queue called **LAB.SEND.AS.XML**.

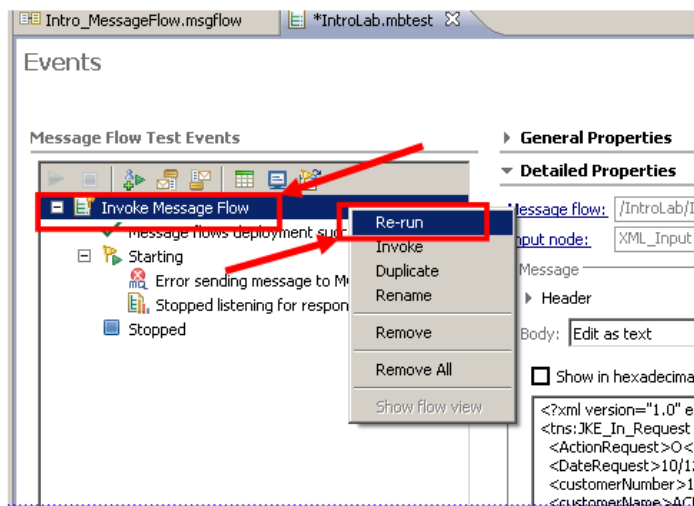
Now we are ready to retry the Test Client.

__26. Switch back to the Test Client in the WebSphere Message Broker Toolkit.

__27. On the left, select the Event called **Invoke Message Flow**.

__28. Press the right mouse button.

__29. Select **Re-run** from the menu.



This time it runs to completion. Recall that the output from our flow was a queue. By selecting the event starting with **MQ Queue Monitor...**, we see the output message. In other words, the Test Client did a “GET” operation off of the output queue and displays the message.

Message Flow Test Events

Invoke Message Flow

Message flows deployment successfully completed

Starting

Error sending message to MQ Queue "LAB.IN"

Stopped listening for response

Stopped

Invoke Message Flow

Message flows deployment successfully completed

Starting

mq

Sending Message to MQ Queue "LAB.IN"

mq

MQ Queue Monitor "LAB.SEND.AS.XML"

Stopped listening for response

Stopped

General Properties

Detailed Properties

Host:localhost

Port:0

Queue manager:MB8QMGR

Queue:LAB.SEND.AS.XML

Message

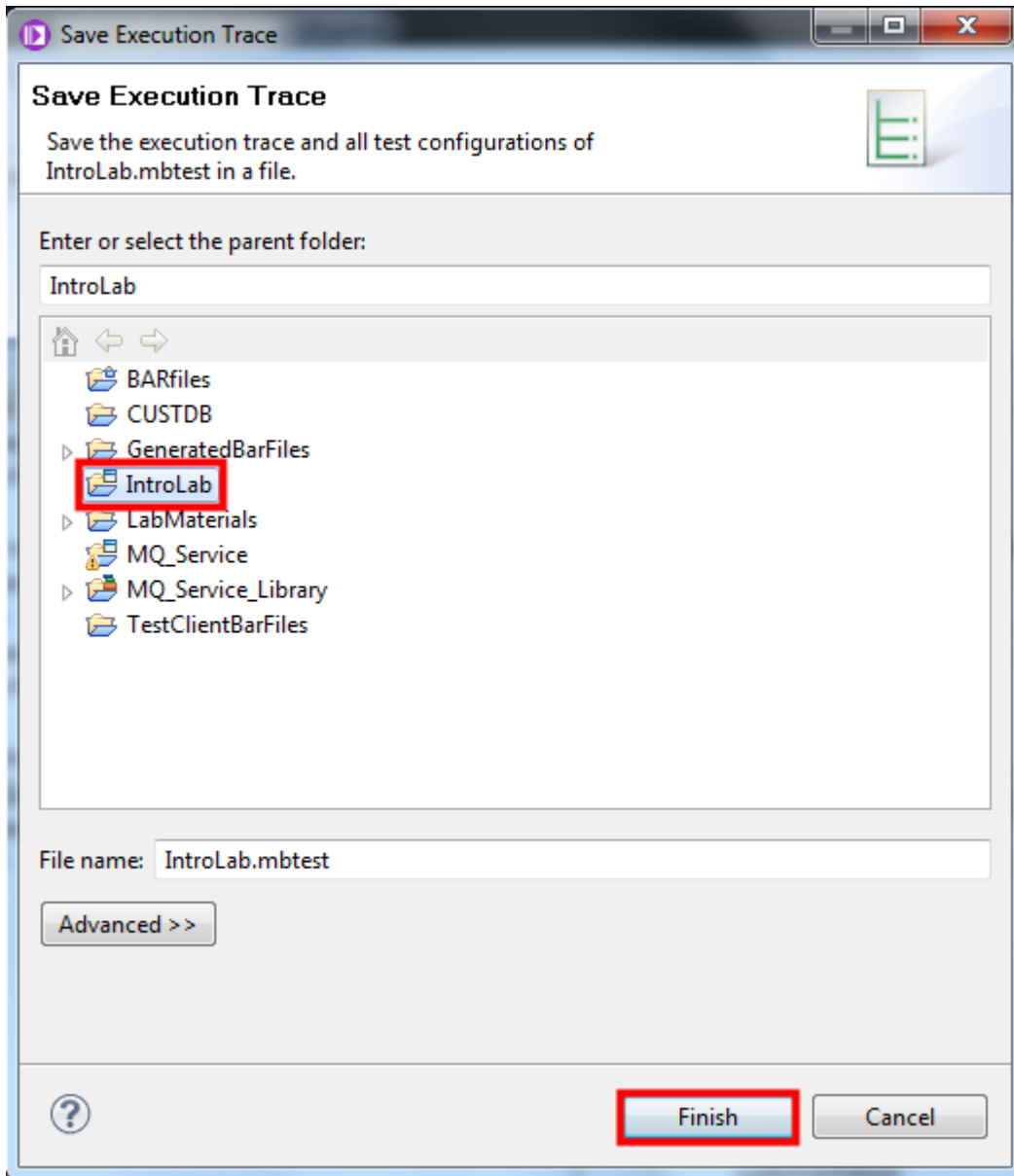
Header

Body:View as XML structure

Name	T...	Value
tns:JKE_In_Request		
xmlns:xsi		http://www.w3.org/2001/XMLSchema-ins...
xmlns:xsd		http://www.w3.org/2001/XMLSchema
xmlns:tns		http://www.ibm.lab.com
ActionRequest		0
DateRequest		10/12/2005
customerNumber		1
customerName		ACME Hardware
customerDetails		
customerAddress1		1254 Main St
customerAddress2		Suite 12
customerCity		Dime Box
customerState		TX
customerCountry		USA
customerPostalCode		76543
customerCreditLimit		1200
customerCreditScore		123
contactDetails		
contactFirstName		Freddy
contactLastName		Bloggs
contactPhoneNumber		555-123-6543
requestDecision		Y
comments		Just a Comment

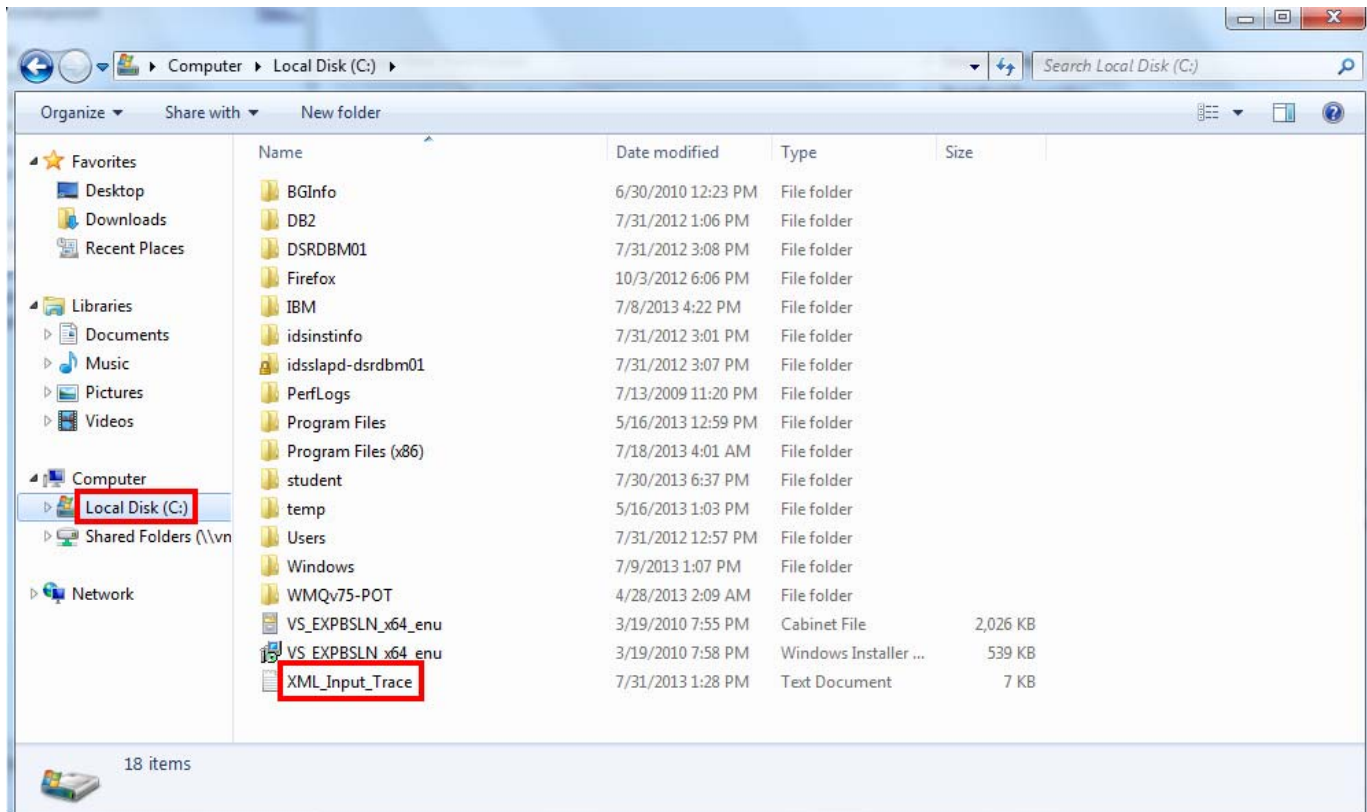


- __30. Select **File→Save** to save the **IntroLab.mbttest** test client configuration.
- __31. In the dialog that appears, select the **IntroLab** application.
- __32. Click **Finish**.



The output of the Trace Node in the file system will be examined next.

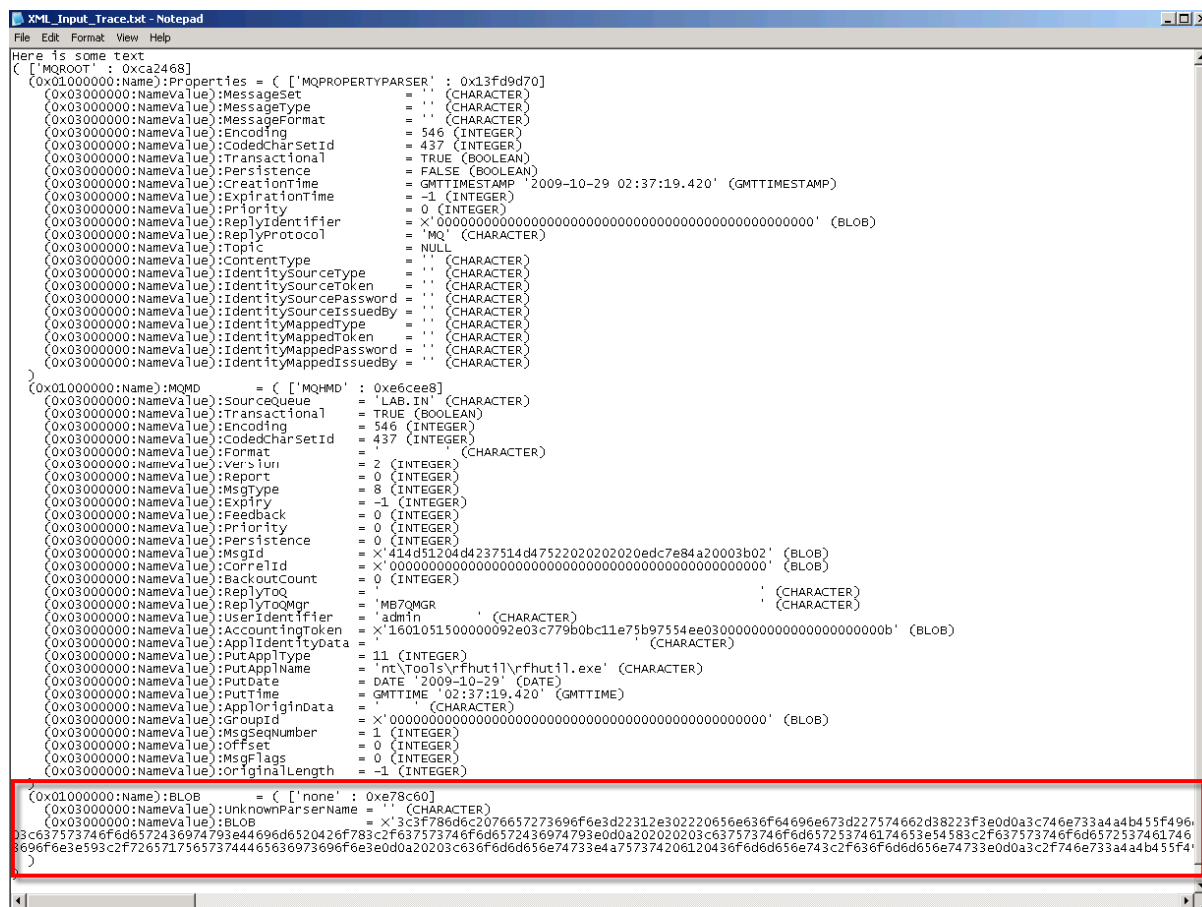
- __33. Bring up Windows Explorer.
- __34. Select the **Local Disk (C:)** folder (root directory).
- __35. Double click **XML_Input_Trace.txt**.



Viewing the contents of the trace file, you can see the line of raw text you configured for the Trace node to display, as well as some detailed information about the message.

__36. To see the actual payload or application data, scroll down to the bottom (**Ctrl+End**).

Hmmmm...what's going on here? The message is a BLOB – a Binary Large Object. Just a string of bytes shown in hexadecimal. What happened to the XML message? Where are the tags? What about the data! All will be resolved soon!



__37. Close the Notepad window.

__38. Minimize the Windows Explorer window.

Each time that you test the message flow new data will be appended to the end of the trace file. You will need to scroll down to the end of the file to see the latest information.

This is the end of Lab 1.

Lab 2 Message models and working with XML messages

2.1 Overview

In this lab, the IntroMessageFlow will be modified to identify the parser (XMLNSC) to be used to process the message.

The steps are very simple.

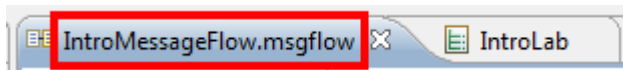
The properties of the Input node will be modified.

The Test Client will be used to run another test.

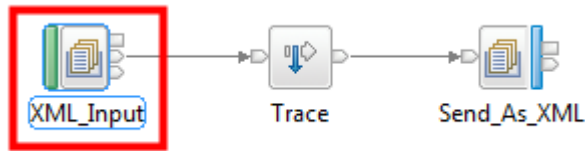
The trace file contents will be viewed to see the difference.

2.2 Using the XML Parser

- __1. Return to the IBM Integration Toolkit.
- __2. Click the **IntroMessageFlow** tab to bring the message flow into view.

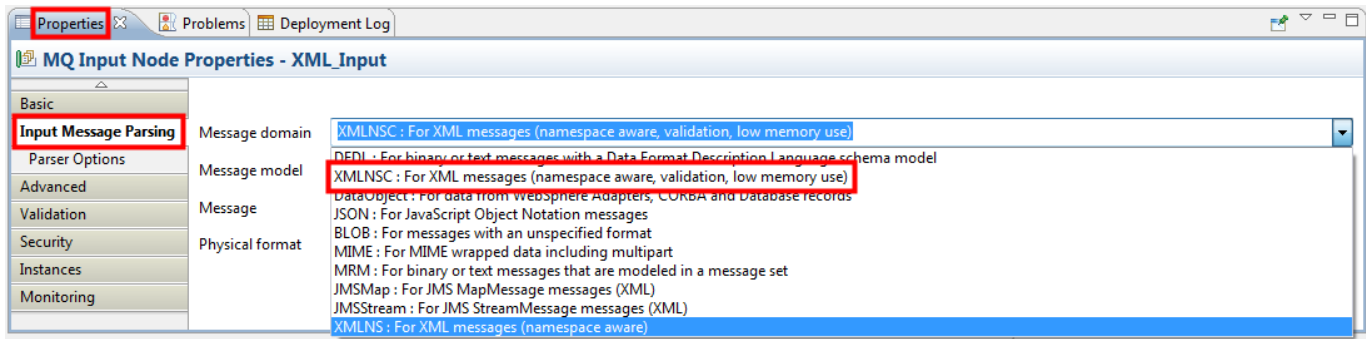



- __3. click the **XML_Input** node to bring its properties into view.



The message flow will be modified so that it uses the XMLNSC parser to process the input message.

- ___4. On the **Properties** view at the bottom of the screen, click the **Input Message Parsing** tab. Since nothing was specified when the node was added, the Message domain (that is, the parser) defaults to binary large object (**BLOB**) – which you saw in the trace.
- ___5. Click the pull-down for the **Message domain**. The various parsers are listed along with a short description. Depending on the Message domain selection, the other fields may be enabled or disabled.
- ___6. Select the **XMLNSC** parser. The **XMLNSC** parser that supports Namespaces (the NS part) and builds a more efficient or compact tree (the C part). The compact tree uses less memory.



- ___7.  Save the message flow (**Ctrl+S**).

Key Idea: Parsers and message domains

IBM Integration Bus supplies a range of parsers to parse and write messages in different formats.

A parser is called when the bit stream that represents an input message must be converted to the format that is used internally by the broker; this process is known as parsing. The input is a bit stream, and the output is a logical tree representation of the message.

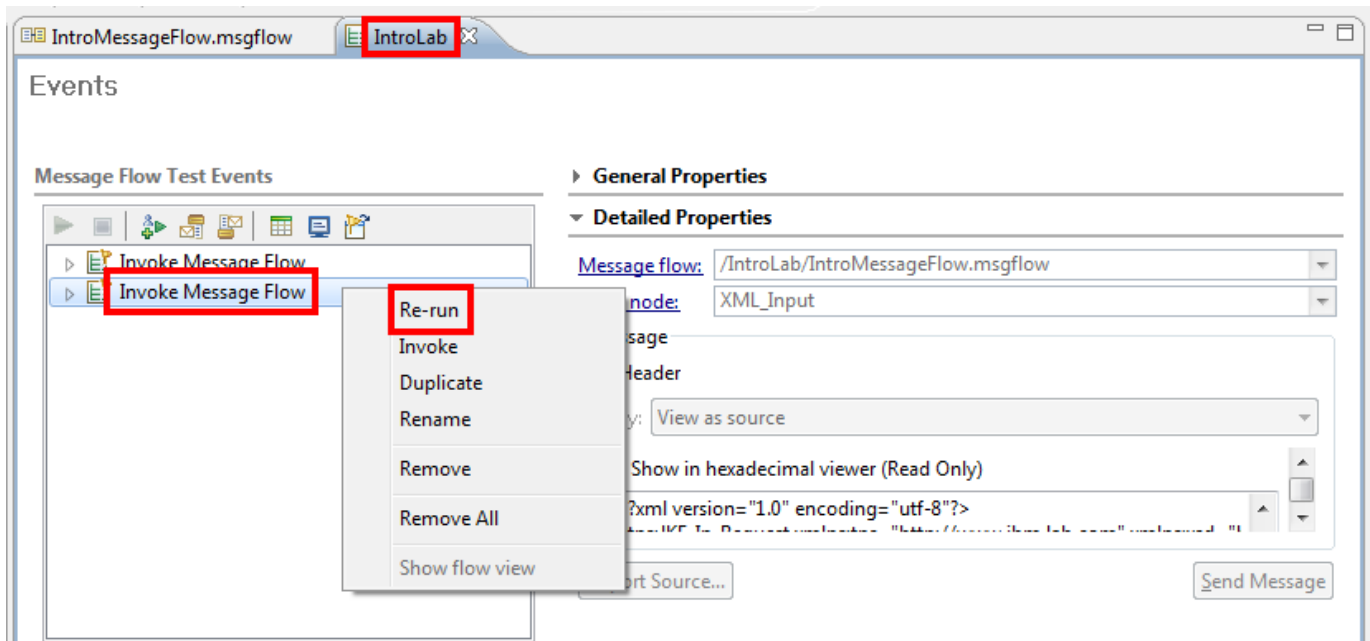
A serializer is called when a logical tree structure must be converted into a bit stream (for example on an Output node). This process is known as serializing.

Each parser is suited to a particular class of messages, known as a message domain. The following list contains some examples of the message domains used in IBM Integration Bus:

- XMLNSC – for XML documents
- DFDL – for general text or binary data streams including industry standards
- JSON – for JSON documents
- DataObject – for data without a stream representation (used by adapters)

Now, let's re-run the Test Client.

- __8. Switch back to the Test Client (**IntroLab** tab).
- __9. Select one of the **Invoke Message Flow** items.
- __10. Press the right mouse button.
- __11. Select **Re-run** from the menu.



You again see the same output message from the Test Client.

The screenshot displays the IBM Integration Bus v9 Test Client interface. On the left, the 'Events' pane shows a list of 'Message Flow Test Events'. The event 'MQ Queue Monitor "LAB_SEND_AS.XML"' is highlighted with a red box. The right pane shows the 'General Properties' and 'Detailed Properties' for the selected event. The 'Detailed Properties' pane is also highlighted with a red box and contains the following information:

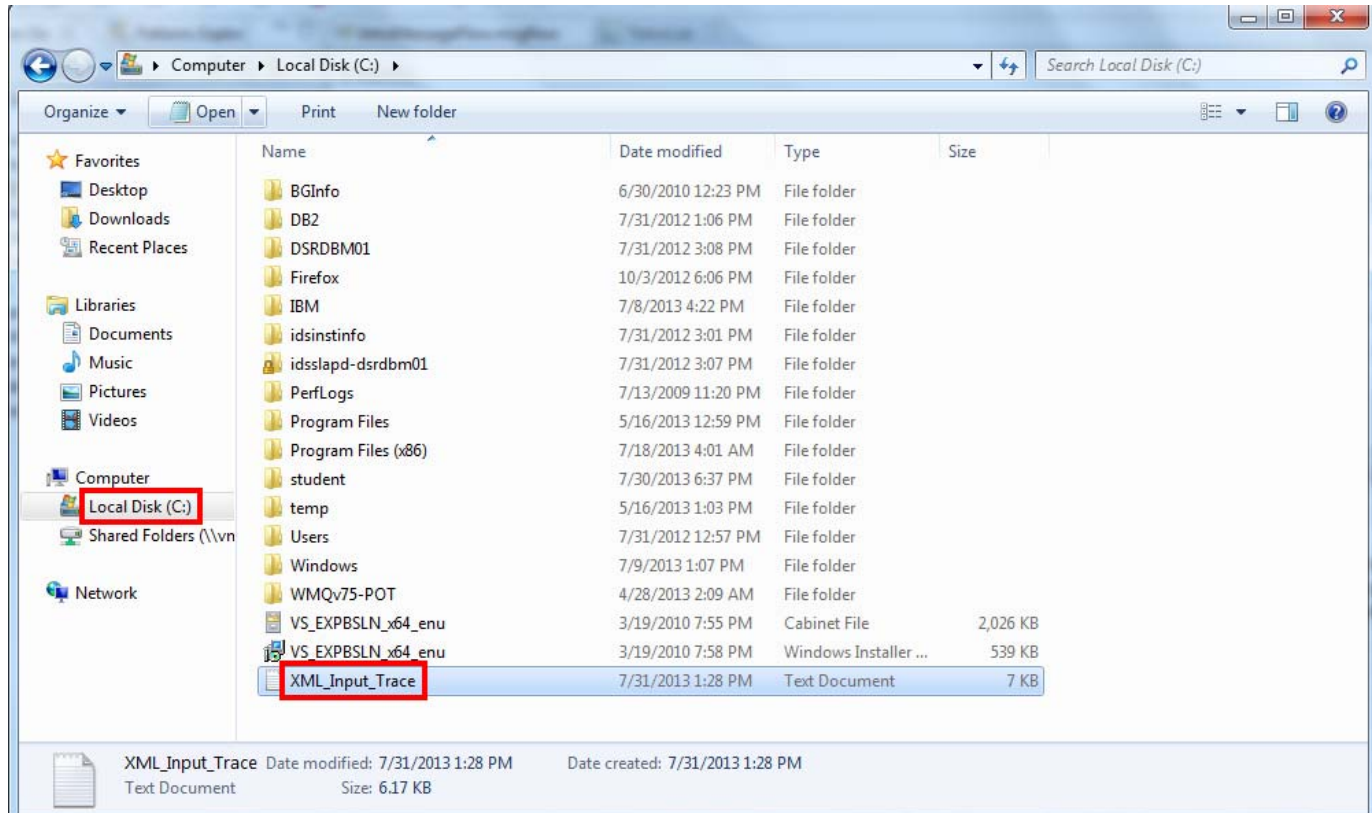
Host: localhost
Port: 0
Queue manager: MBSQMGR
Queue: LAB_SEND_AS.XML

Message
Header
Body: View as XML structure

Name	Value
tns:JKE_In_Request	
xmlns:xsi	http://www.w3.org/2001/XMLSchema-ins...
xmlns:xsd	http://www.w3.org/2001/XMLSchema
xmlns:tns	http://www.ibm.lab.com
ActionRequest	0
DateRequest	10/12/2005
customerNumber	1
customerName	ACME Hardware
customerDetails	
customerAddress	1254 Main St
customerAddress	Suite 12
customerCity	Dime Box
customerState	TX
customerCountry	USA
customerPostalCo	76543
customerCreditLir	1200
customerCreditSc	123
contactDetails	
contactFirstName	Freddy
contactLastName	Bloos

This is only what the output message looks like *after* it arrives on the output queue. Let's see what the message looked like *while* it was passing through the message flow.

- __12. Return to Windows Explorer.
- __13. Navigate to the file at **C:\XML_Input_Trace.txt**.
- __14. Double click **XML_Input_Trace.txt** file.



___15. Scroll to the end of the file (**Ctrl + End**).

Trace output is placed at the end of any existing content in a file, so scroll down to the bottom of the file and view the results. Much more pleasing...here is a nicely formatted message tree that will allow you to conveniently access the fields in the XML message by name. Notice:

- The **XMLNSC** Domain name (which is what we set on the input node).
- All of the element types are String represented by the (CHARACTER)! Why is that? The answer requires understanding both Parsers and Message Models.

```

0x00000000:nameValue):AppIdentityData = 11 (INTEGER)
0x00000000:nameValue):PutAppType = "MyTool\MyFhut\1v\Fhut1.exe" (CHARACTER)
0x00000000:nameValue):PutAppName = "DATE 2009-10-29" (DATE)
0x00000000:nameValue):PutDate = GMTIME 16122108_060 (GMTTIME)
0x00000000:nameValue):PutTime = (CHARACTER)
0x00000000:nameValue):AppLogInData = x'0000000000000000000000000000000000000000000000000000' (Blob)
0x00000000:nameValue):MsgSeqNumber = 1 (INTEGER)
0x00000000:nameValue):MsgSeq = 0 (INTEGER)
0x00000000:nameValue):MsgFlags = 0 (INTEGER)
0x00000000:nameValue):OriginalLength = 1 (INTEGER)
0x00000000:Folder):XMLNSC = ["xmlns": "0xca37d0"]
0x00000000:Attribute):xmlLocation = {
0x00000000:Attribute):version = "1.0" (CHARACTER)
0x00000000:Attribute):encoding = "utf-8" (CHARACTER)
0x00000000:Folder):http://www.ibm.tib.com:KCE-InRequest = {
0x00000000:NameSpaceDocId):http://www.w3.org/2000/xmlns:/tns = "http://www.ibm.tib.com:" (CHARACTER)
0x00000000:NameSpaceDocId):http://www.w3.org/2000/xmlns:/xsd = "http://www.w3.org/2000/XMLSchema" (CHARACTER)
0x00000000:NameSpaceDocId):http://www.w3.org/2000/xmlns:/xsdt = "http://www.w3.org/2000/XMLSchema-instance" (CHARACTER)
0x00000000:PCdataField):dateRequest = "10/12/2005" (CHARACTER)
0x00000000:PCdataField):customerNumber = "1" (CHARACTER)
0x00000000:PCdataField):customerName = "ACME Hardware" (CHARACTER)
0x00000000:Folder):customerDetails = {
0x00000000:PCdataField):customerAddress1 = "1234 Main" (CHARACTER)
0x00000000:PCdataField):customerAddress2 = "Suite 12" (CHARACTER)
0x00000000:PCdataField):customerCity = "Dime Box" (CHARACTER)
0x00000000:PCdataField):customerState = "TX" (CHARACTER)
0x00000000:PCdataField):customerCountry = "USA" (CHARACTER)
0x00000000:PCdataField):customerPostalCode = "76543" (CHARACTER)
0x00000000:PCdataField):customerCreditLimit = "1200" (CHARACTER)
0x00000000:PCdataField):customerCreditScore = "123" (CHARACTER)
}
0x00000000:Folder):contactDetails = {
0x00000000:PCdataField):contactFirstName = "Freddy" (CHARACTER)
0x00000000:PCdataField):contactLastName = "Bloggs" (CHARACTER)
0x00000000:PCdataField):contactPhoneNumber = "555-123-6543" (CHARACTER)
}
{
0x00000000:PCdataField):requestDescription = "Y" (CHARACTER)
0x00000000:PCdataField):comments = "Just a comment" (CHARACTER)
}

```

___16. Close the Notepad window.

__17. Minimize Windows Explorer.

2.3 Creating a Message Model from an XSD

In this portion of the lab, we will use a message model to parse the XML message.

Key Idea: Message Models

Much of the business world relies on the exchange of information between applications. This information is contained in messages that have a defined structure that is known and agreed to by the sender and the receiver.

Applications typically use a combination of message formats, including those message formats that are defined by the following structures or standards:

- Comma Separated Values (CSV)
- COBOL, C, PL1, and other language data structures
- Industry standards such as SWIFT, X12 or HL7
- XML including SOAP

You can model a wide variety of message formats so that they can be understood by IBM Integration Bus message flows. When the message format is known, the broker can parse an incoming message bit stream and convert it into a logical message tree for manipulation by a message flow.

Some message formats are self-defining and can be parsed without reference to a model. However, most message formats are not self-defining, and a parser must have access to a predefined model that describes the message if it is to parse the message correctly.

An example of a self-defining message format is XML. In XML, the message itself contains metadata in addition to data values, and it is this metadata that enables an XML parser to understand an XML message even if no model is available. Another example of a self-defining format is JSON.

Examples of messages that do not have a self-defining message format are CSV text messages, binary messages that originate from a COBOL program, and SWIFT formatted text messages. None of these message formats contain sufficient information to enable a parser to fully understand the message. In these cases, a model is required to describe them.

Even if your messages are self-defining, and do not require modeling, message modeling has the following advantages:

- Runtime validation of messages. Without a message model, a parser cannot check whether input and output messages have the correct structure and data values.
- Enhanced parsing of XML messages. Although XML is self-defining, all data values are treated as strings if a message model is not used. If a message model is used, the parser is provided with the data type of data values, and can cast the data accordingly.
- Code completion assistance when coding transformation. When you are creating ESQL programs for your message flows, the ESQL editor can use message models to provide code completion assistance.
- Graphical mapping. Without message models, you cannot use the Message Mapping editor.
- Reuse of message models, in whole or in part, by creating additional messages that are based on existing messages.
- Generation of documentation.

- Provision of version control and access control for message models by storing them in a central repository.

Message models allow the full use of the facilities that are offered by IBM Integration Bus.

To speed up the creation of message models, importers are provided to read metadata such as C header files, COBOL copybooks, and EIS (Enterprise Information System, such as SAP) metadata, and to create message models from that metadata. Additionally, predefined models are available for common industry standard message formats such as SWIFT, EDIFACT, X12, FIX, HL7, and TLOG.

The XML Parser was run in *programmatic* mode where it parsed the XML message, so it assumed everything was a string. By parsing with a model, we can get a message with typed elements and one that is subject to constraints (such as required fields, maximum field lengths and so on). The toolkit provides wizards to import your existing models (such as WSDLs, XSDs, copybooks and others.)

- ___1. In the Application Development view (project navigator) on the left, right click the white space.
- ___2. Select **New→Message Model...** from the menu.



- __3. Select the **Other XML** radio button (under **XML**).
- __4. Check out some of the other options for which there are import wizards.
- __5. Click **Next**.

New Message Model

Create a new message model file

Select the message model type or format

XML

- ☐ **SOAP XML** XML data for use in Web Services.
- ☒ **Other XML** All other XML data.

Text and binary

- ☐ **CSV text** Comma Separated Values data, a delimited text format commonly used as an export format by spreadsheets and databases.
- ☐ **Record-oriented text** Text data formats where delimited fields are grouped into records.
- ☐ **COBOL** Data for COBOL programs
- ☐ **C** Data for C programs
- ☐ **Other text or binary** All other text or binary data formats.

Enterprise Information Systems

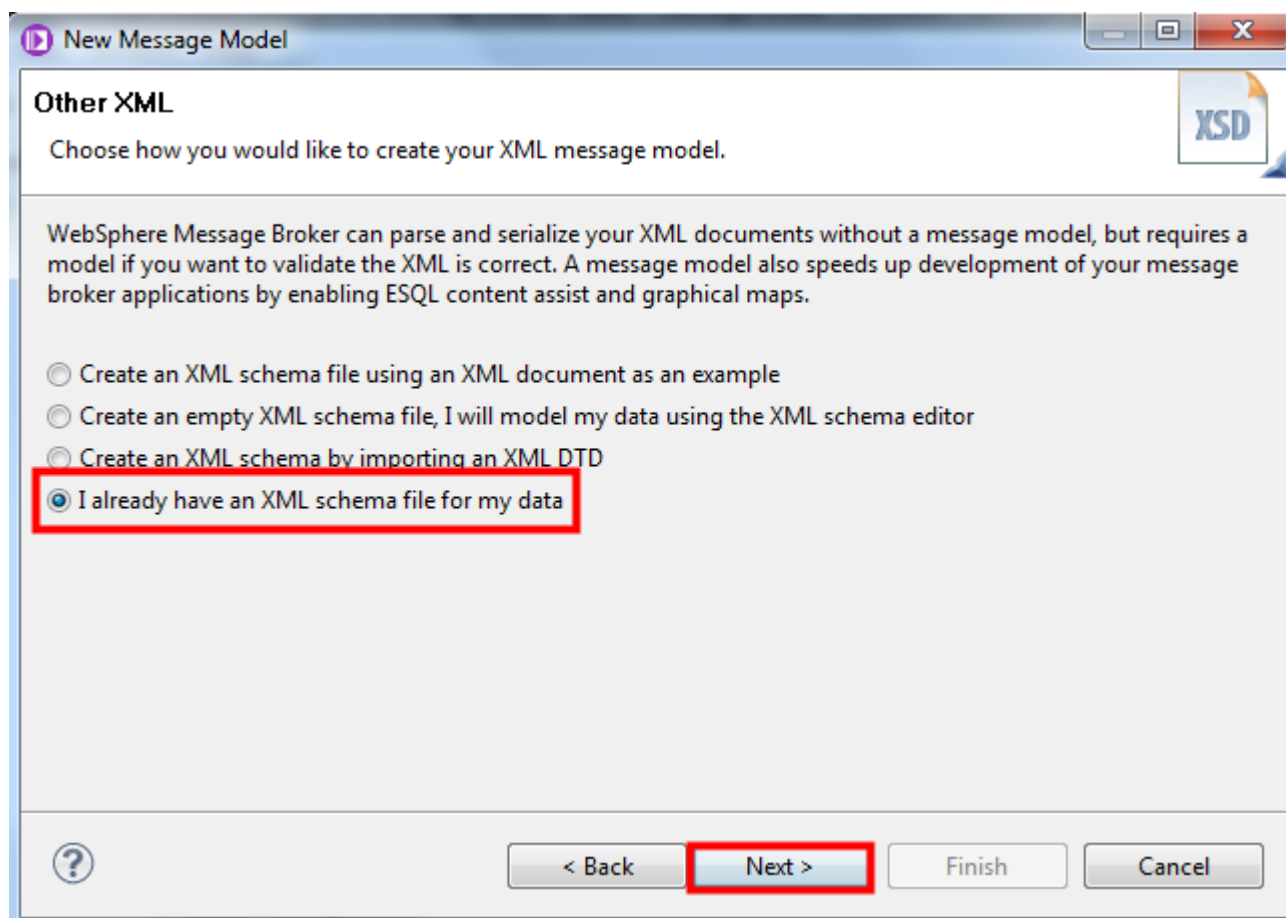
- ☐ **SAP** Data from SAP systems including IDoc and BAPI
- ☐ **Siebel** Data from Siebel systems
- ☐ **PeopleSoft** Data from PeopleSoft
- ☐ **JD Edwards** Data from JD Edwards systems

Other

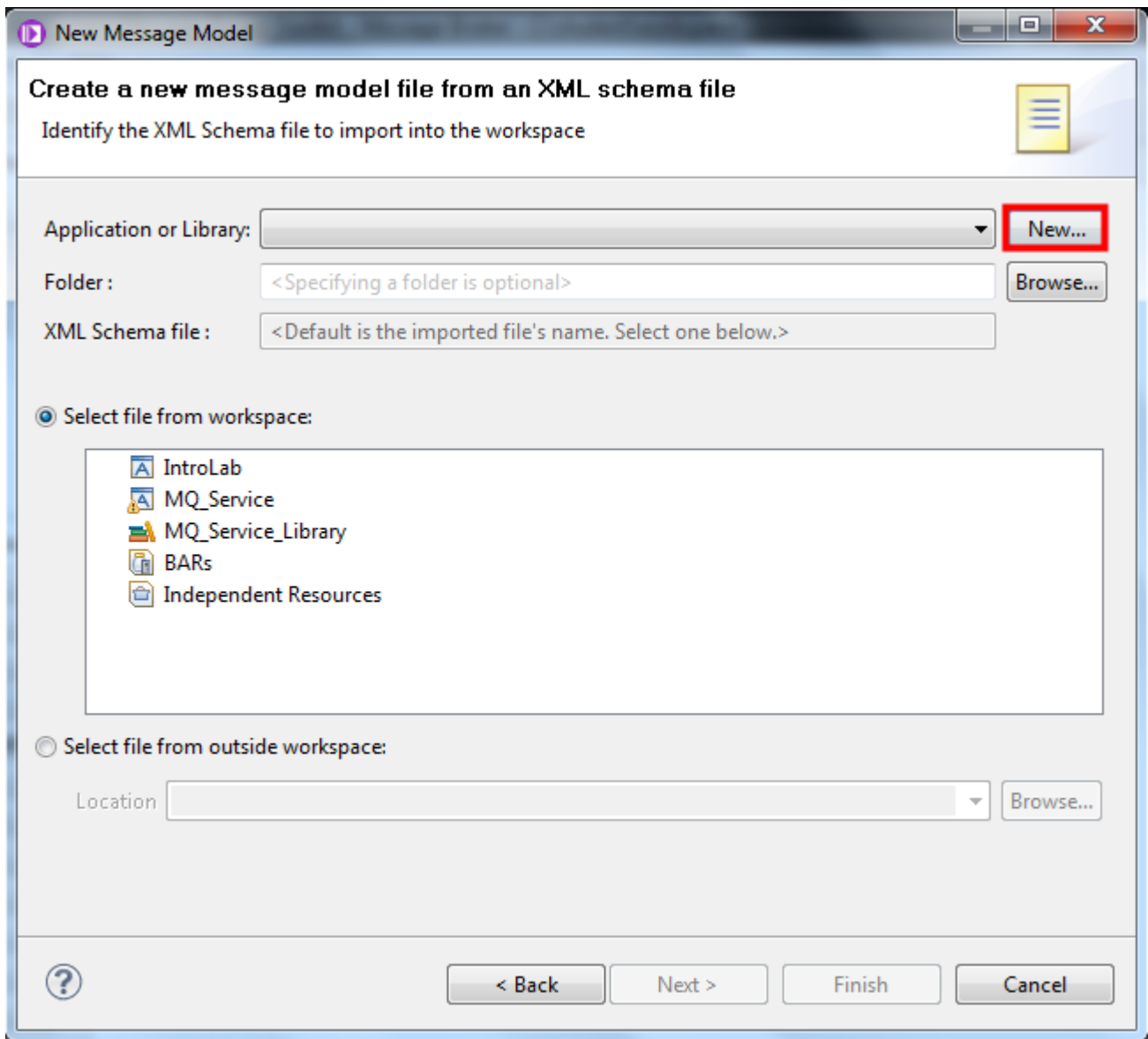
- ☐ **CORBA IDL** Data from CORBA
- ☐ **Database record** Records from relational databases
- ☐ **MIME** Data for extended email format
- ☐ **IBM supplied** Predefined data format

Navigation: ? < Back **Next >** Finish Cancel

- __6. Select the **I already have an XML schema for my data** radio button.
- __7. Click **Next**.

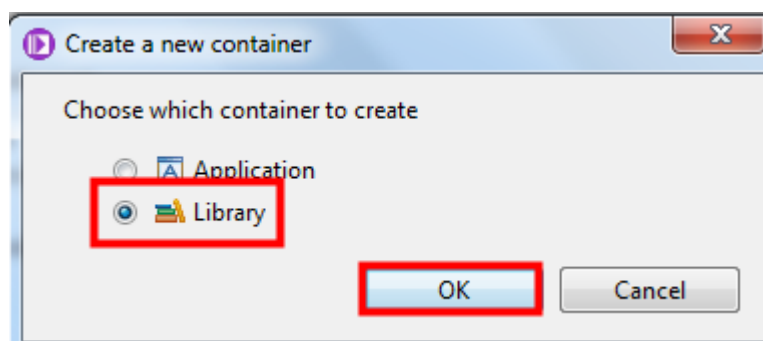


__8. Click the **New..** button next to the **Application or Library** field.



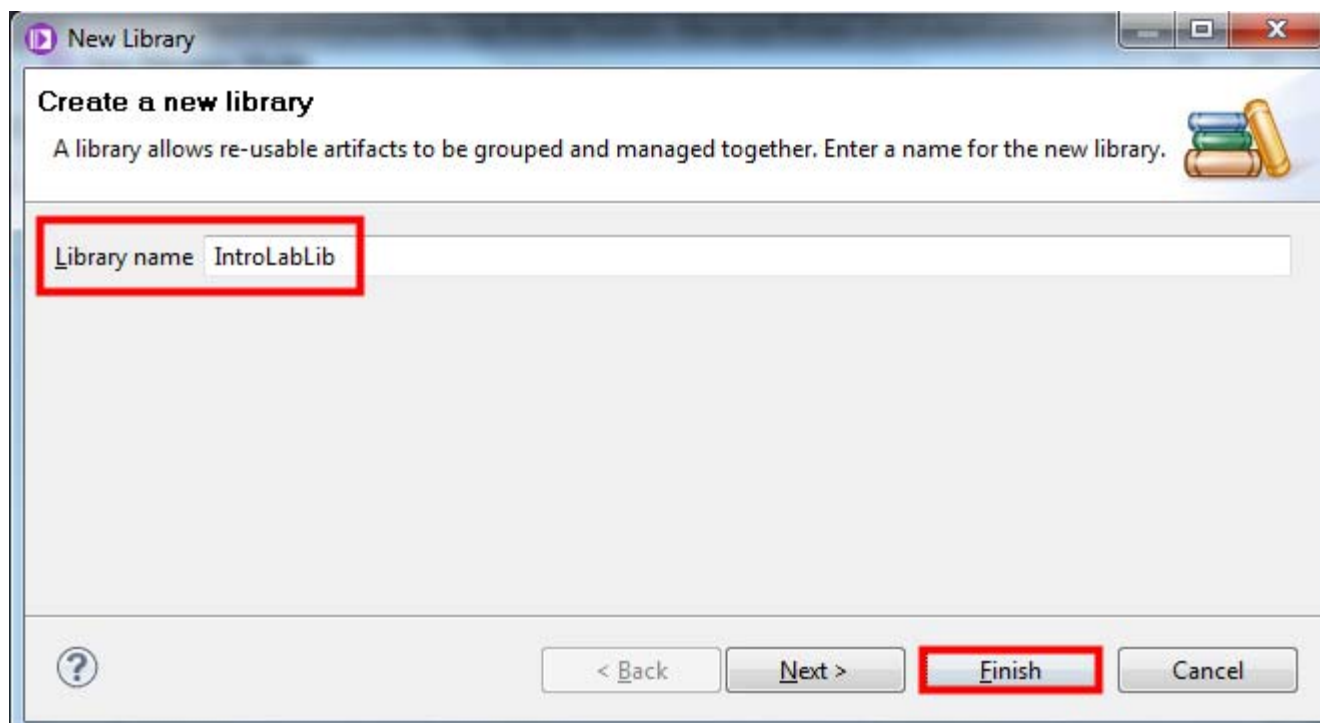
__9. In the popup dialog, select **Library**.

__10. Click **OK**.

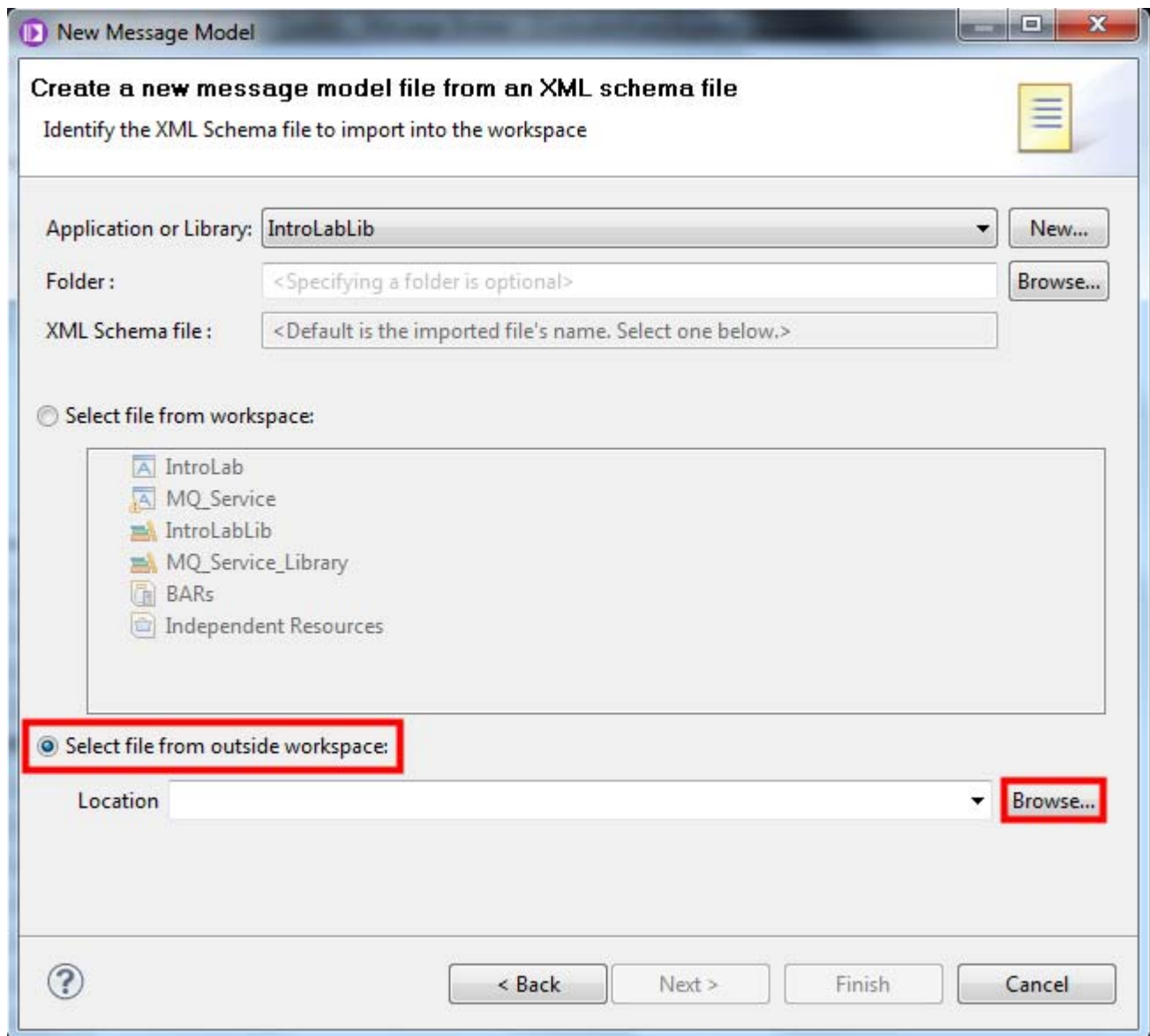


__11. In the popup dialog, type **IntroLabLib** as the **Library name**.

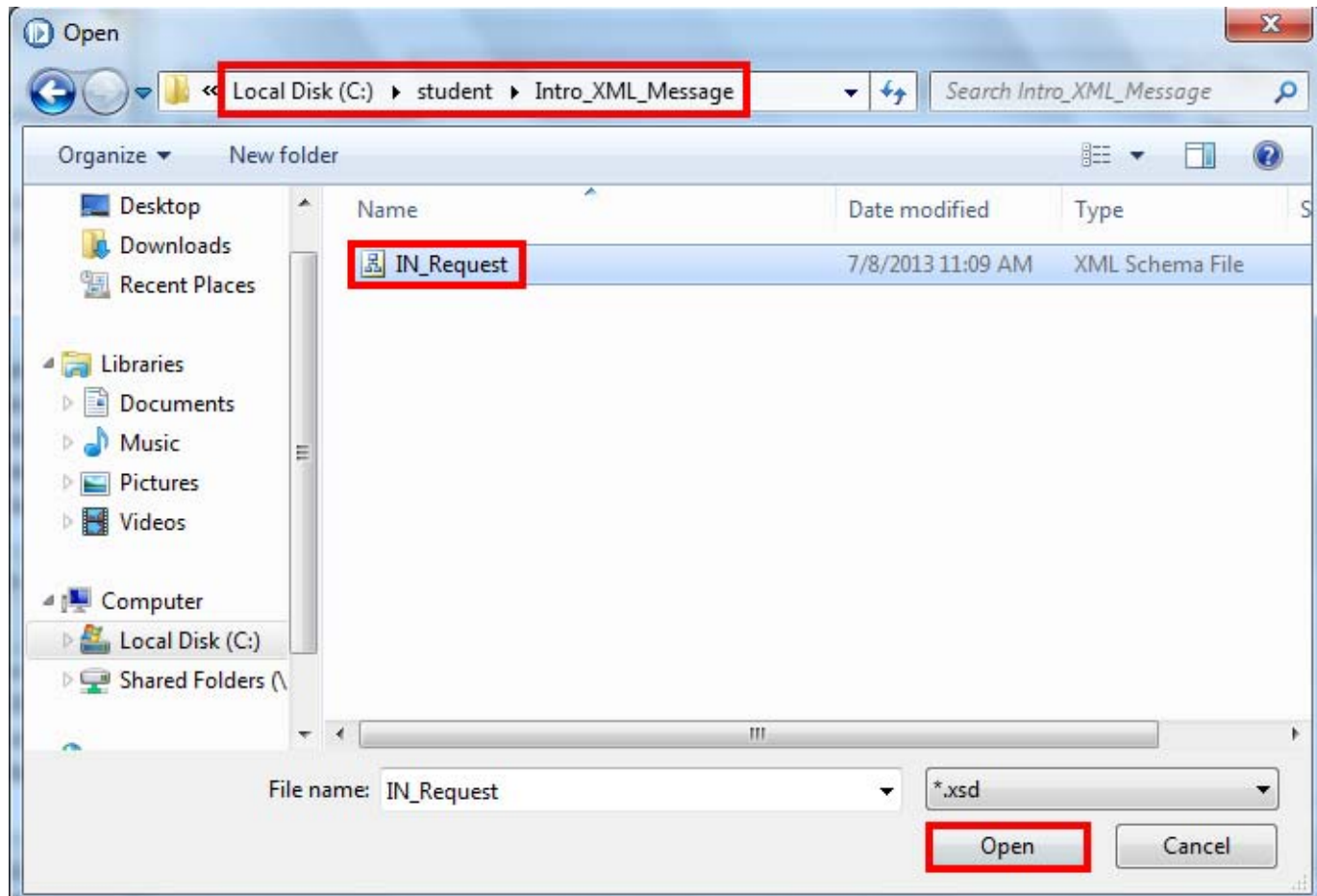
__12. Click **Finish**.



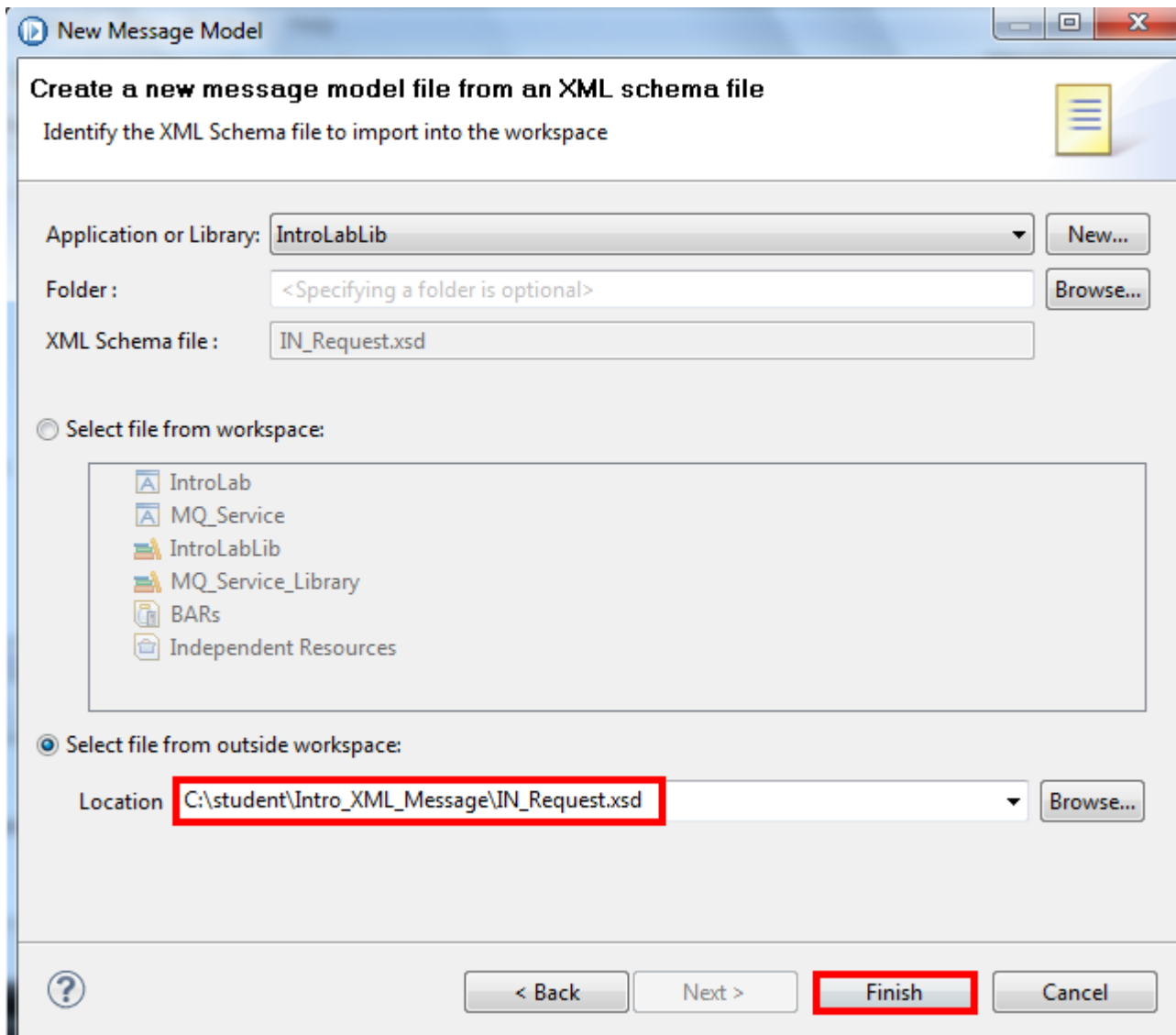
- __13. Back in the Message Model wizard, check the radio button **Select file from outside workspace**.
- __14. Click **Browse...**



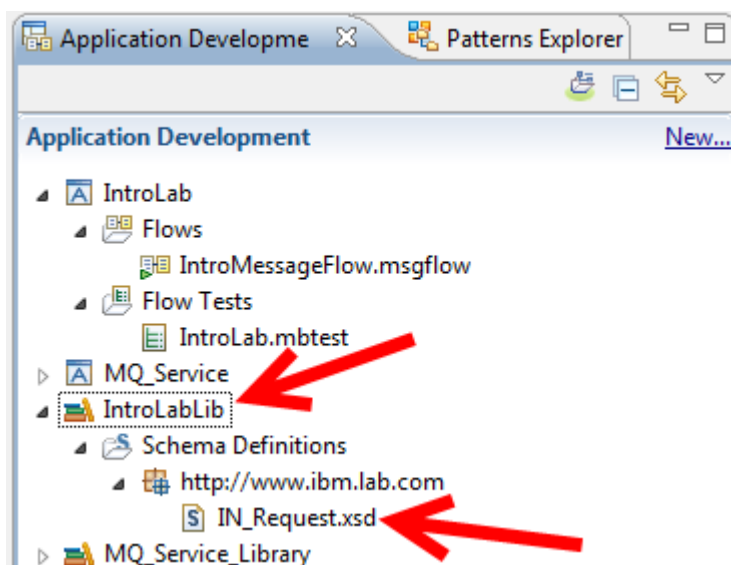
- __15. Navigate to **C:\Student\Intro_XML_Message** folder.
- __16. Select **IN_Request.xsd**.
- __17. Click **Open**.



__18. Back in the Message Model Wizard, click **Finish**.



You now have a Library project with the XML message model for the input message.



Key Idea: Library Projects

Applications and libraries are deployable containers of resources, such as message flows, message definitions (DFDL, XSD files), JAR files, XSL style sheets, and WebSphere Adapters files.

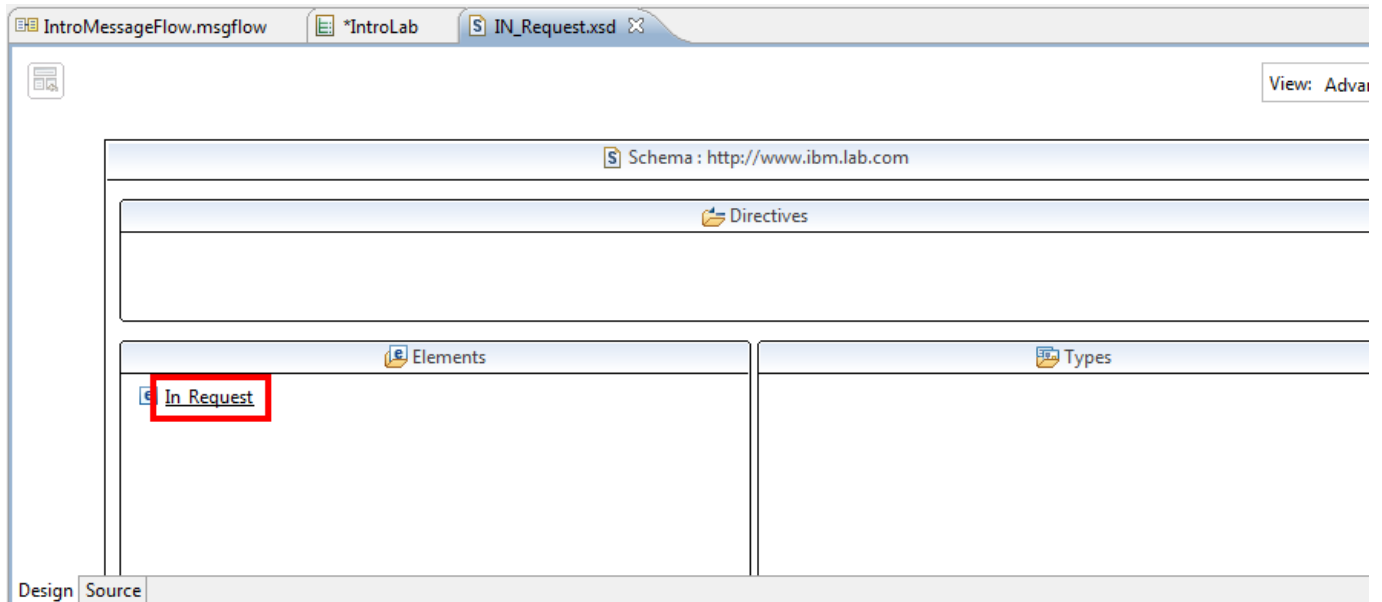
A library is a logical grouping of related code, data, or both. A library contains references to reusable resources, such as a message model or map. A library can refer to a resource that is contained in another library. Libraries are optional. They are typically used to reuse resources. Libraries can be either embedded in an application (private) or obtained by a message flow (that is not part of an application) dynamically at run time (execution group level). Use multiple libraries to group related resources (for example, by type or function).

Consider using libraries if you want to share routines and definitions with multiple teams, projects, or brokers. Libraries are also useful if you need to use different versions of a coherent set of routines and definitions.

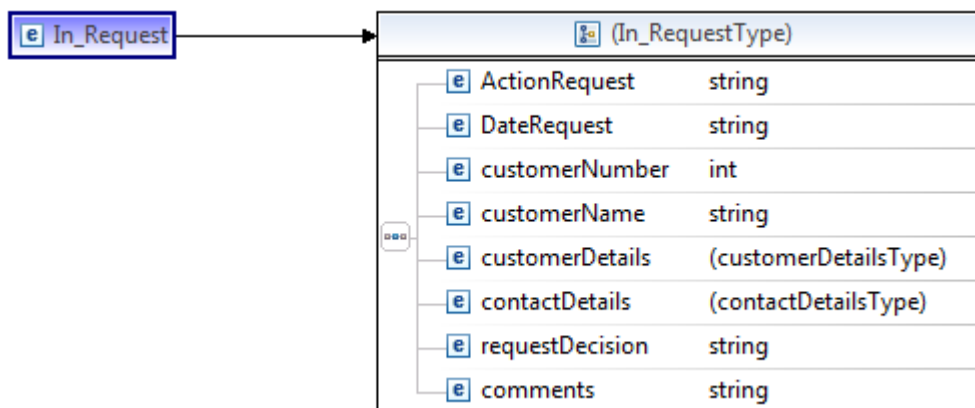
Using a library is typically not necessary if you do not need to regularly reuse IBM Integration Bus routines or definitions.

Notice that the XSD is opened for you after import and is visible using the XML Schema Editor which shows you both a GUI representation of your XML schema as well as the source. **In_Request** is the only Global element. If you double click it, you can drill down into its structure.

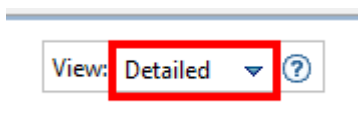
__19. Double click the **In_Request** element to view the message elements.



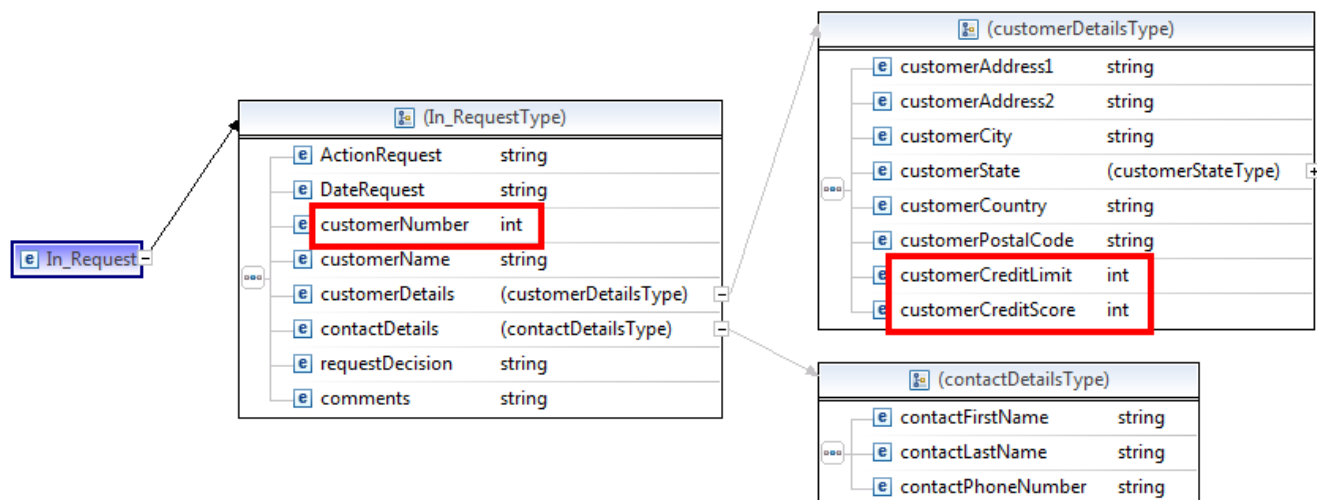
The message model should now be visible.



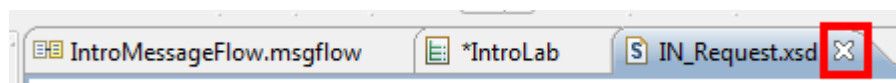
__20. Select the **Detailed** view.



Some elements, such as `customerNumber` and `customerCreditScore`, are integers (ints) and not strings.



__21. Close the **IN_Request.xsd** tab.

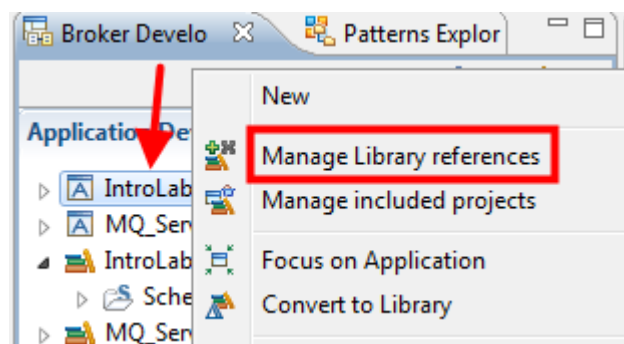


Now, let's update the message flow to use the message model when parsing incoming messages.

__22. In the project view on the left, select the **IntroLab** application.

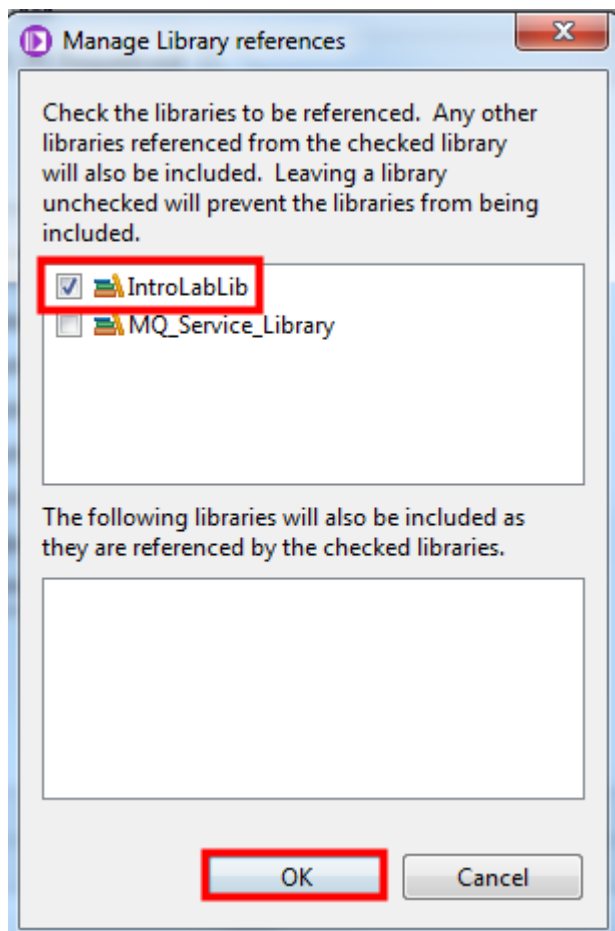
__23. Press the right mouse button.

__24. Select **Manage Library references**.



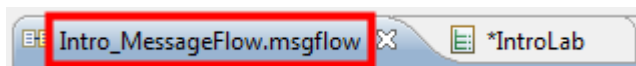
__25. Select the **IntroLabLib** check box.

__26. Click **OK**.



We need to tell the parser to run in *schema-driven* mode, rather than operate in *programmatic* mode.

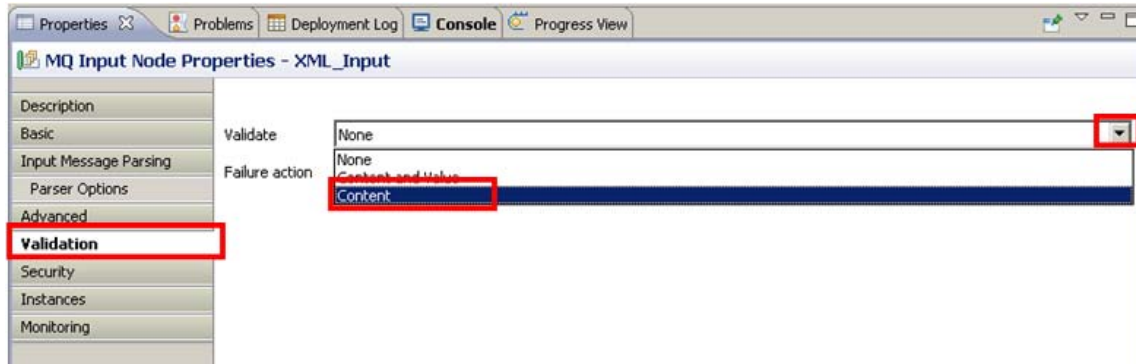
__27. Click the **Intro_MessageFlow.msgflow** tab to return to the message flow editor.



__28. Single click the **XML_Input** node, in order to edit its properties.

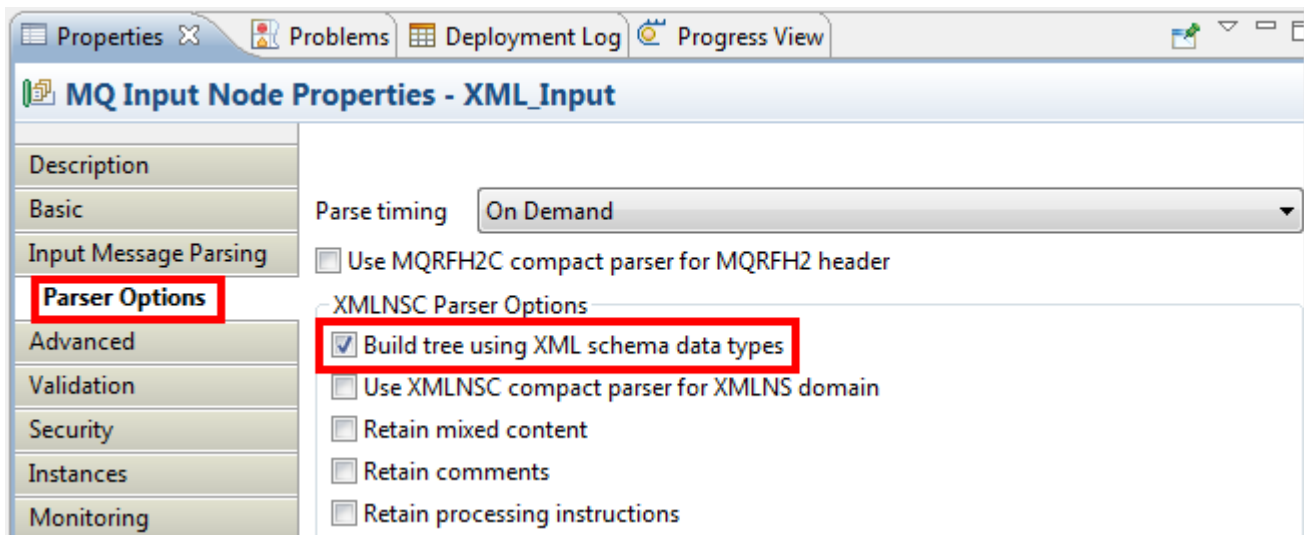
__29. In the **Properties** view, click the **Validation** tab.


__30. In the **Validation** dropdown, select **Content**.



__31. Select the **Parser Options** tab.

__32. Select the **Build tree using XML schema data types** check box.

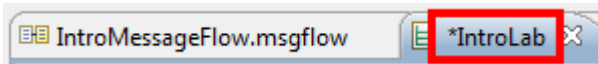


__33.  Save the message flow (**Ctrl+S**).

2.4 Re-running the Test Client

The flow will now be run again. The trace output will then be examined.

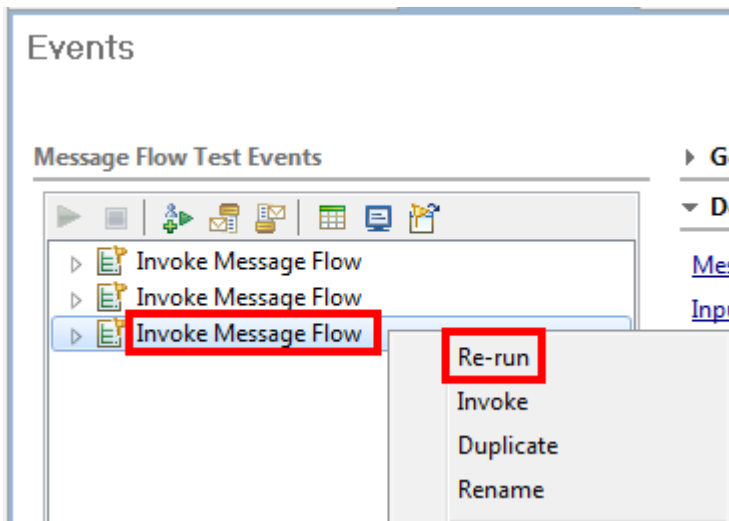
- ___1. In the editor, select the **IntroLab.mbtest** tab (or reopen from the App in the navigator).



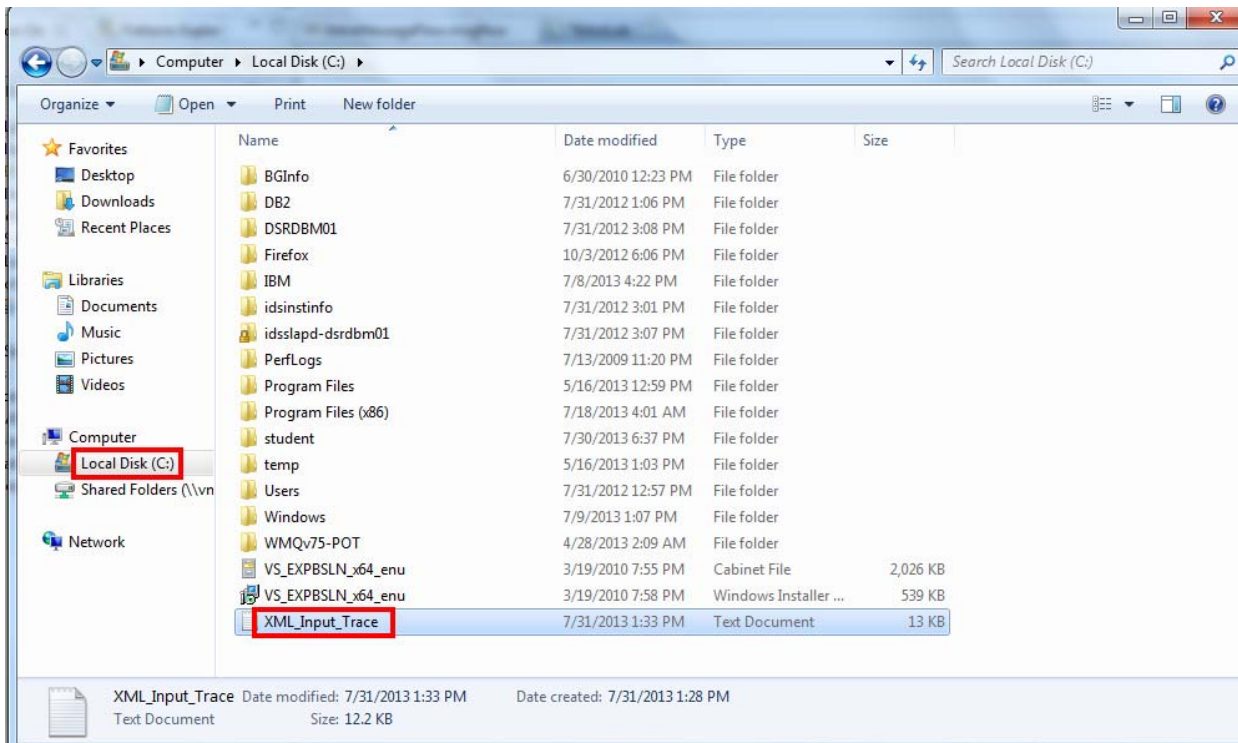
- ___2. Right click **Invoke Message Flow**.

- ___3. Click **Re-run**.

The tooling will automatically rebuild and redeploy the App with the dependent Library included.



- __4. Return to the Windows Explorer window.
- __5. Double click the **C:\XML_Input_Trace.txt** file.



- __6. Scroll down to the end of the file (or use **Ctl+End**), and view the new trace output.

```
File Edit Format View Help
(0x03000100:Attribute):Version = '1.0' (CHARACTER)
(0x03000100:Attribute):Encoding = 'utf-8' (CHARACTER)
)
(0x01000000:Folder)http://www.ibm.lab.com:JKE_In_Request = (
(0x03000102:NamespaceDef)http://www.w3.org/2000/xmlns:/tns = 'http://www.ibm.lab.com' (CHARACTER)
(0x03000102:NamespaceDef)http://www.w3.org/2000/xmlns:/xsd = 'http://www.w3.org/2001/XMLSchema' (CHARACTER)
(0x03000102:NamespaceDef)http://www.w3.org/2000/xmlns:/xsi = 'http://www.w3.org/2001/XMLSchema-instance' (CHARACTER)
(0x03000000:PCDataField):ActionRequest = 'o' (CHARACTER)
(0x03000000:PCDataField):DateRequest = '10/12/2005' (CHARACTER)
(0x03000000:PCDataField):customerNumber = 1 (INTEGER)
(0x03000000:PCDataField):customerName = 'ACME Hardware' (CHARACTER)
(0x01000000:Folder):customerDetails = (
(0x03000000:PCDataField):customerAddress1 = '1254 Main St' (CHARACTER)
(0x03000000:PCDataField):customerAddress2 = 'Suite 12' (CHARACTER)
(0x03000000:PCDataField):customerCity = 'Dime Box' (CHARACTER)
(0x03000000:PCDataField):customerState = 'TX' (CHARACTER)
(0x03000000:PCDataField):customerCountry = 'US' (CHARACTER)
(0x03000000:PCDataField):customerPostalCode = '76543' (CHARACTER)
(0x03000000:PCDataField):customerCreditLimit = 1200 (INTEGER)
(0x03000000:PCDataField):customerCreditScore = 123 (INTEGER)
)
(0x01000000:Folder):contactDetails = (
(0x03000000:PCDataField):contactFirstName = 'Freddy' (CHARACTER)
(0x03000000:PCDataField):contactLastName = 'Bloggs' (CHARACTER)
(0x03000000:PCDataField):contactPhoneNumber = '555-123-6543' (CHARACTER)
)
(0x03000000:PCDataField):requestDecision = 'Y' (CHARACTER)
(0x03000000:PCDataField):comments = 'Just a Comment' (CHARACTER)
)
)
```

- __7. Close the Notepad window.
- __8. Minimize Windows Explorer.

This is the end of lab 2.

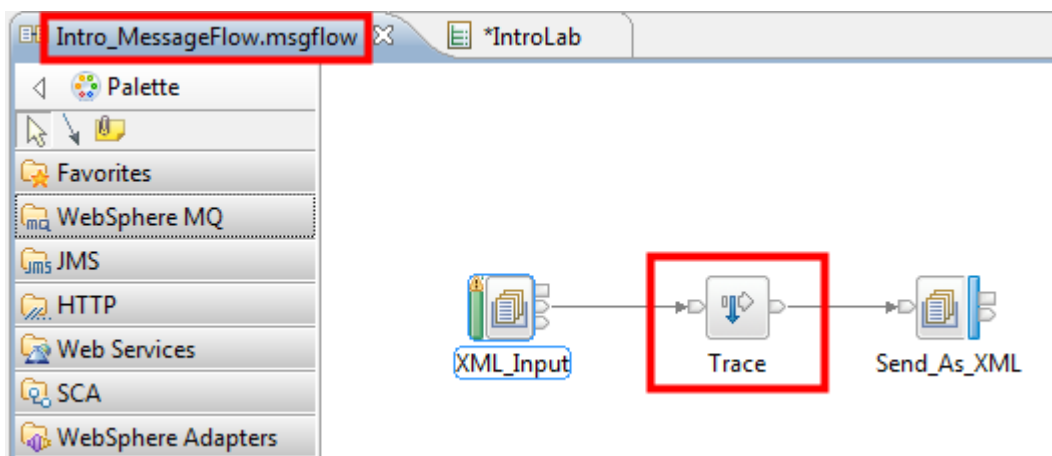
Lab 3 Content-based routing and the Debugger

3.1 Overview

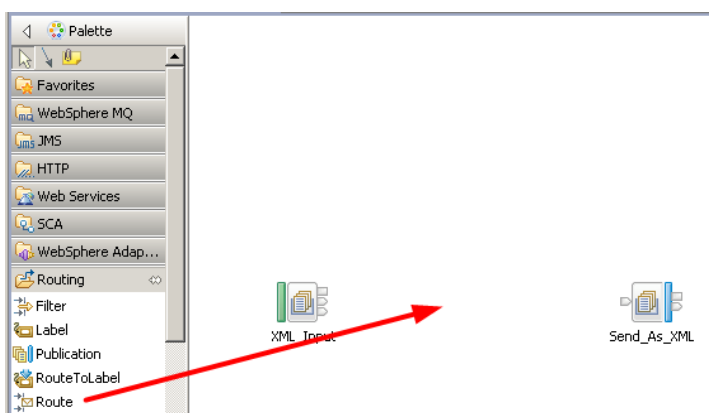
In this lab, you will perform simple routing. Input messages will be sent to one of three destinations depending on the country code. Addresses in the United States will be sent to a US shipping queue, while addresses in Canada will be sent to a Canadian shipping queue, and those that are not in the United States or Canada will be sent to a third queue.

3.2 Add routing logic

- __1. Return to the Integration Bus Toolkit.
- __2. Select the **IntroMessageFlow** message flow in the **message flow editor**.
- __3. Select the **Trace** node.
- __4. Press the **Delete** key.



- __5. Expand the **Routing** drawer in the **Palette**.
- __6. Select a **Route** node and place it between the **XML Input** and **Send_As_XML** nodes.



__7. Change the name of the new routing node to **CheckCountry**.

__8. Press the **Enter** key to complete the rename operation.



__9. Wire the **Out** terminal of the **XML_Input** node to the **In** terminal of the **CheckCountry** node.

__10. Wire the **Default** terminal of the **CheckCountry** node to the **Send_As_XML** node.

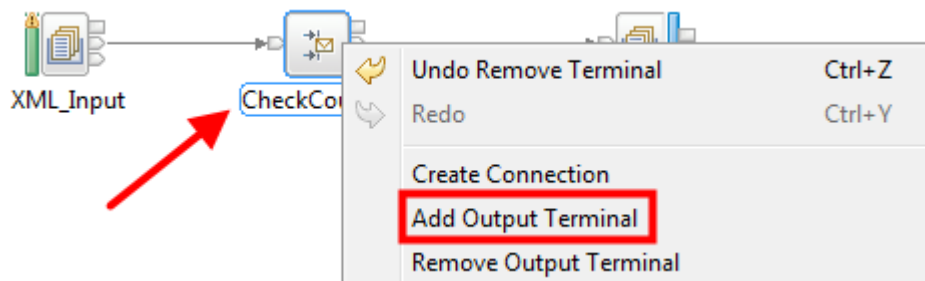


Terminals will now be added to the **CheckCountry** routing node for **US** and **Canadian** addresses.

__11. Select the **CheckCountry** route node.

__12. Press the right mouse button.

__13. Select **Add Output Terminal** from the menu.



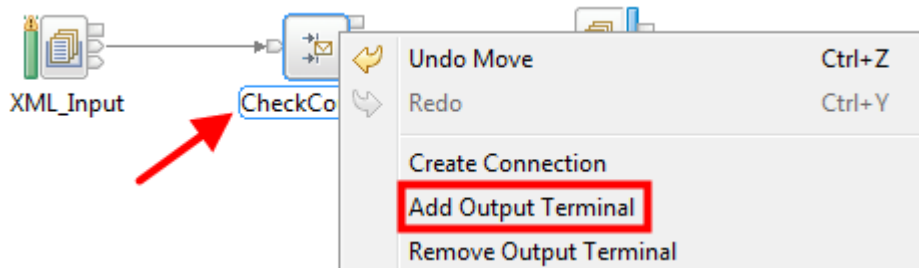
__14. Enter **US** as the name of the new output terminal.

__15. Press the **OK** button to continue.

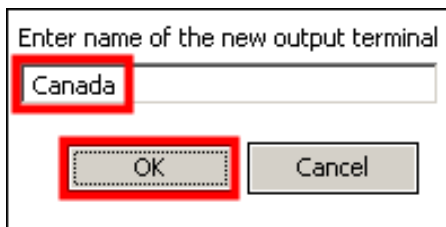
The screenshot shows a dialog box titled 'Enter name of the new output terminal'. The input field contains the text 'US'. Below the input field are two buttons: 'OK' and 'Cancel'. The 'OK' button is highlighted with a red rectangular box.

The steps will now be repeated to add a terminal called **Canada**.

- __16. Select the **CheckCountry** node.
- __17. Press the right mouse button.
- __18. Select **Add Output Terminal** from the menu.

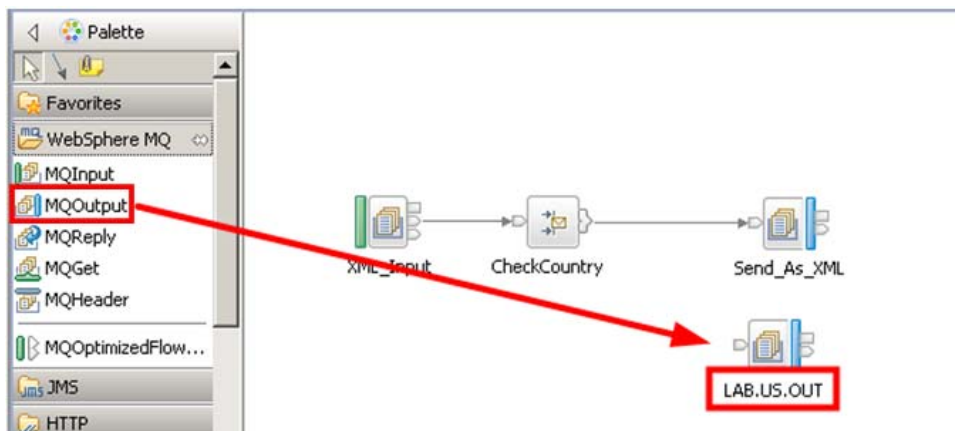


- __19. Enter **Canada** as the name of the new output terminal.
- __20. Press the **OK** button to continue.

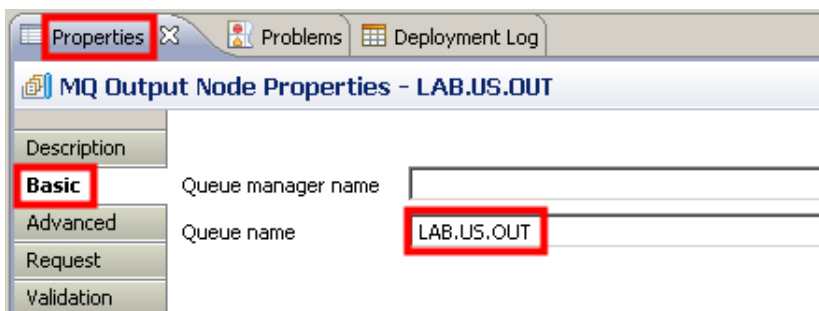


Two additional output destinations will now be added. One will be for addresses within the United States and a second one will be for addresses within Canada. All other addresses will be sent to the existing output queue.

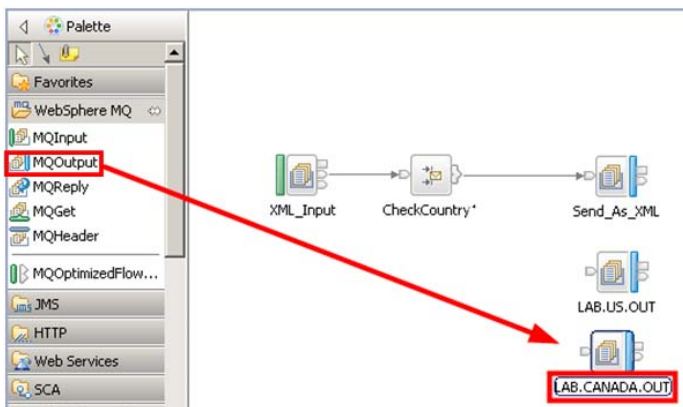
- __21. On the **Palette** expand the **WebSphere MQ** drawer.
- __22. Select an **MQOutput** node.
- __23. Place the new node below the **Send_As_XML** node.
- __24. Change the name of the node to **LAB.US.OUT**
- __25. Press the **Enter** key to complete the rename operation.



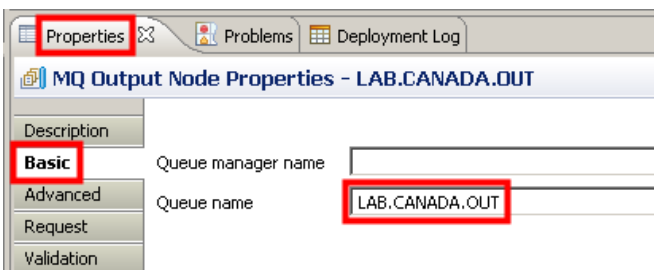
- __26. In the **Properties** pane select the **Basic** tab.
- __27. Enter **LAB.US.OUT** as the **Queue name** parameter.



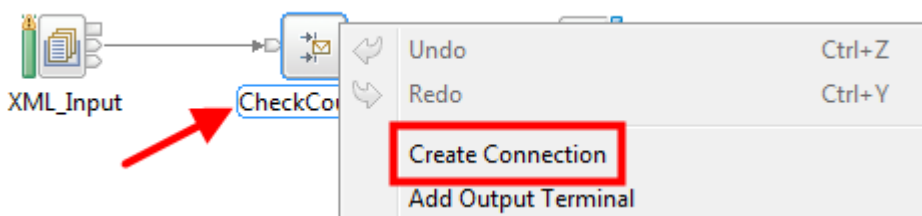
- __28. Select the **MQOutput** node in the **WebSphere MQ** drawer again.
- __29. Move the mouse pointer to a point below the **LAB.US.OUT** node.
- __30. Press the left mouse button to place the new node below the **LAB.US.OUT** node.
- __31. Change the name to **LAB.CANADA.OUT**.
- __32. Press the **Enter** key to complete the rename operation.



- __33. In the **Properties** pane select the **Basic** tab.
- __34. Enter **LAB.CANADA.OUT** as the **Queue name** parameter.

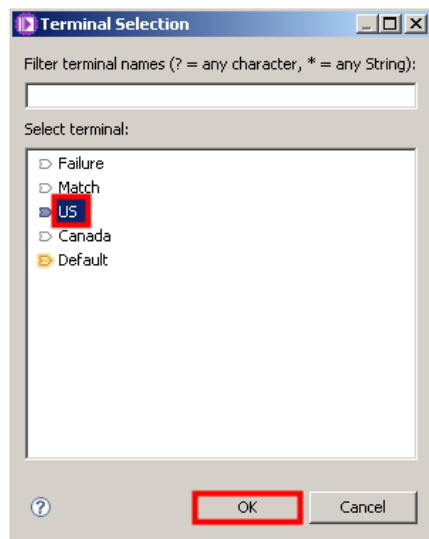


- __35. Right-click the **CheckCountry** route node.
- __36. Select **Create Connection** from the menu.



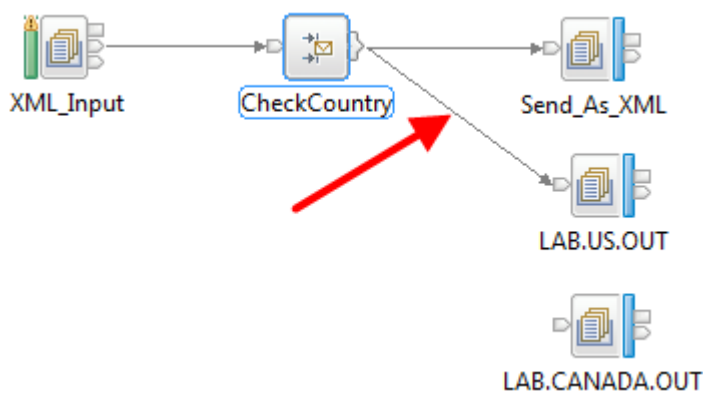
__37. In the Terminal Selection popup, select **US**.

__38. Click **OK**.



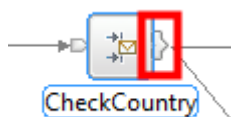
__39. Drag the wire that appears to the **LAB.US.OUT** node.

__40. Press the left mouse button to complete the connection.



__41. Another way to make a connection is just to click the terminal itself. Try this by selecting the output terminal of the **CheckCountry** node.

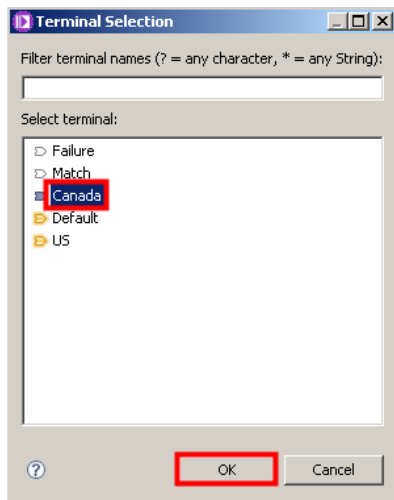
__42. Press the left mouse button.



Because there are too many terminals to individually depict graphically, a dialog box appears allowing you to select the proper terminal.

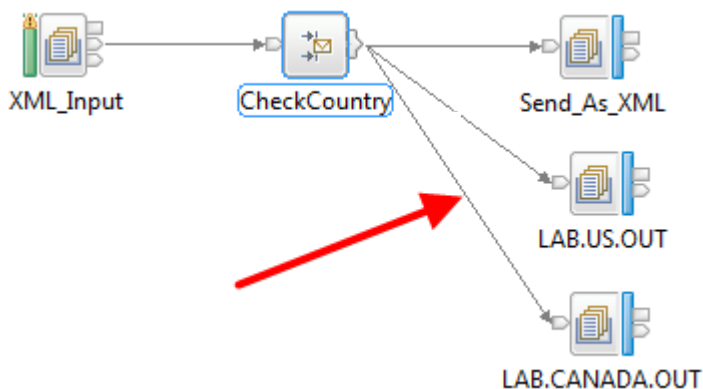
__43. Select the **Canada** terminal.

__44. Press the **OK** button.



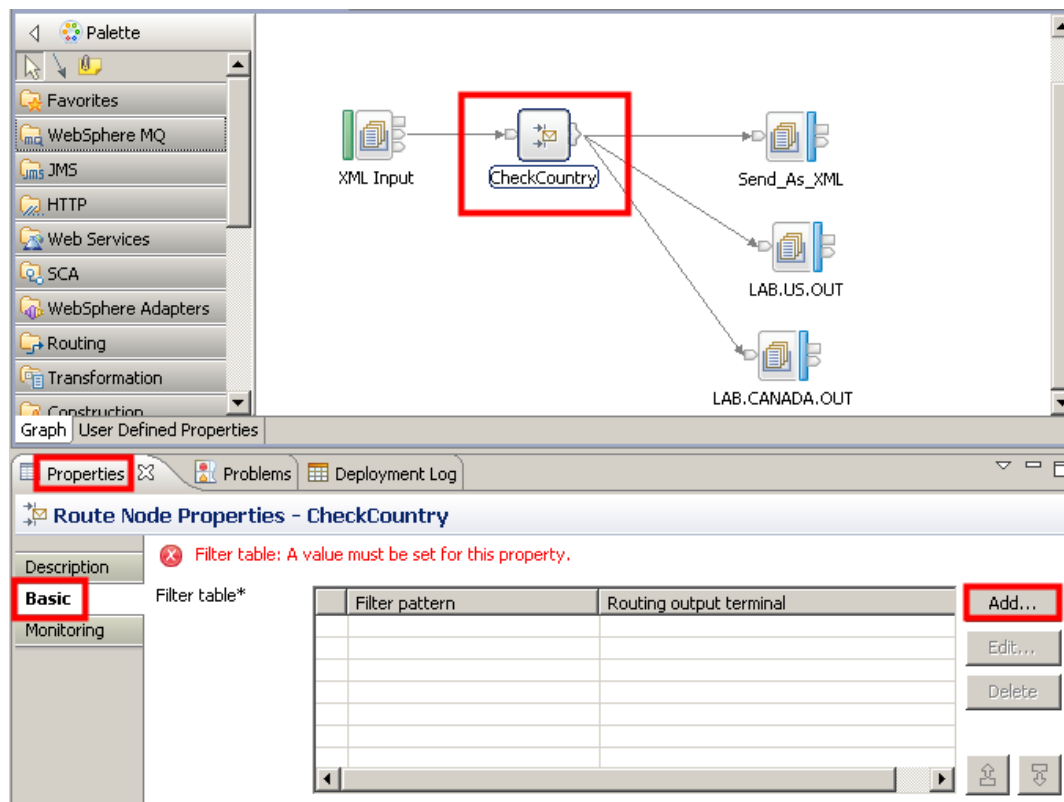
__45. Drag the wire that appears to the **LAB.CANADA.OUT** node.

__46. Press the left mouse button to complete the connection.

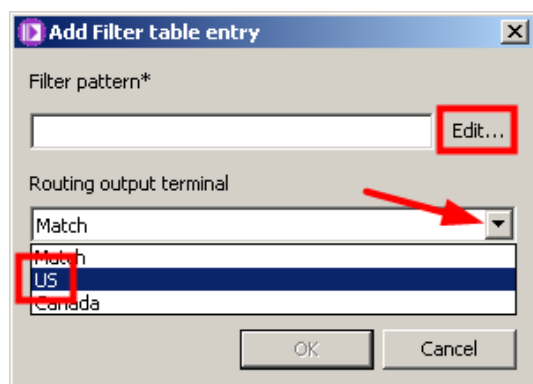


The criteria to be used by the **CheckCountry** routing node must now be specified.

- __47. Select the **CheckCountry** node.
- __48. In the **Properties** pane, select the **Basic** tab.
- __49. Press the **Add** button.

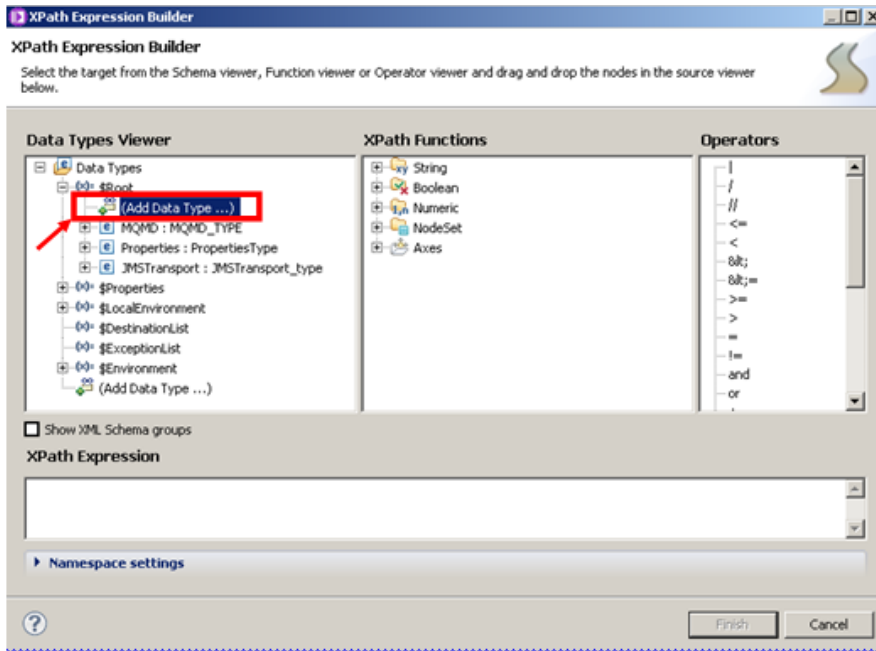


- __50. Use the drop-down menu to select the **US** terminal as the **Routing output terminal**.
- __51. Press the **Edit...** button.



__52. Expand **\$Root**.

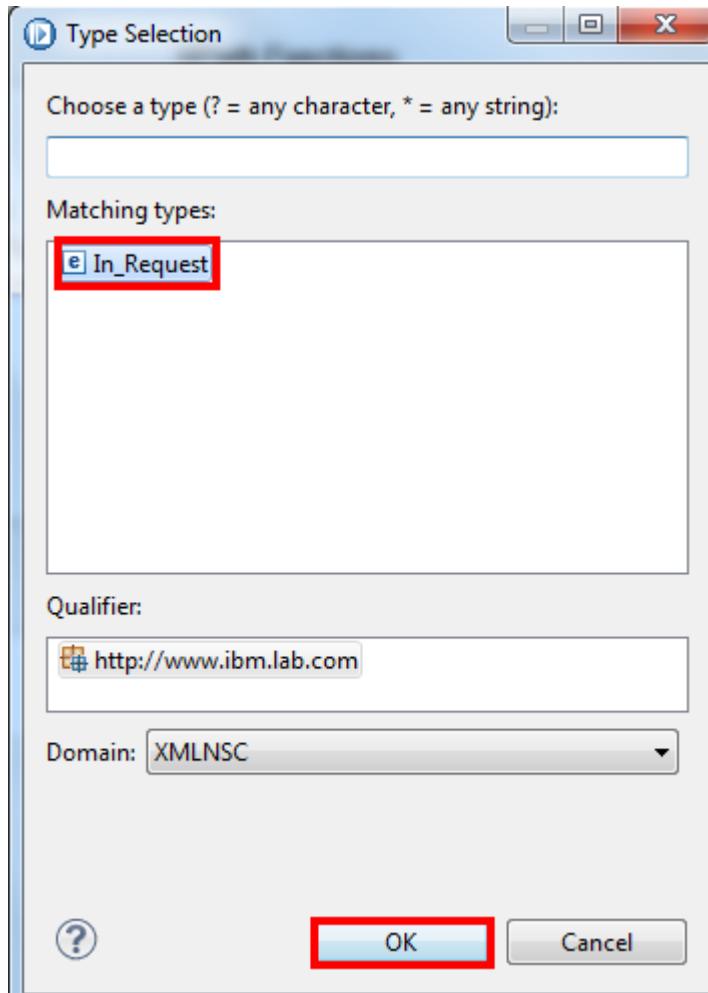
__53. Select **(Add Data Type ...)**.



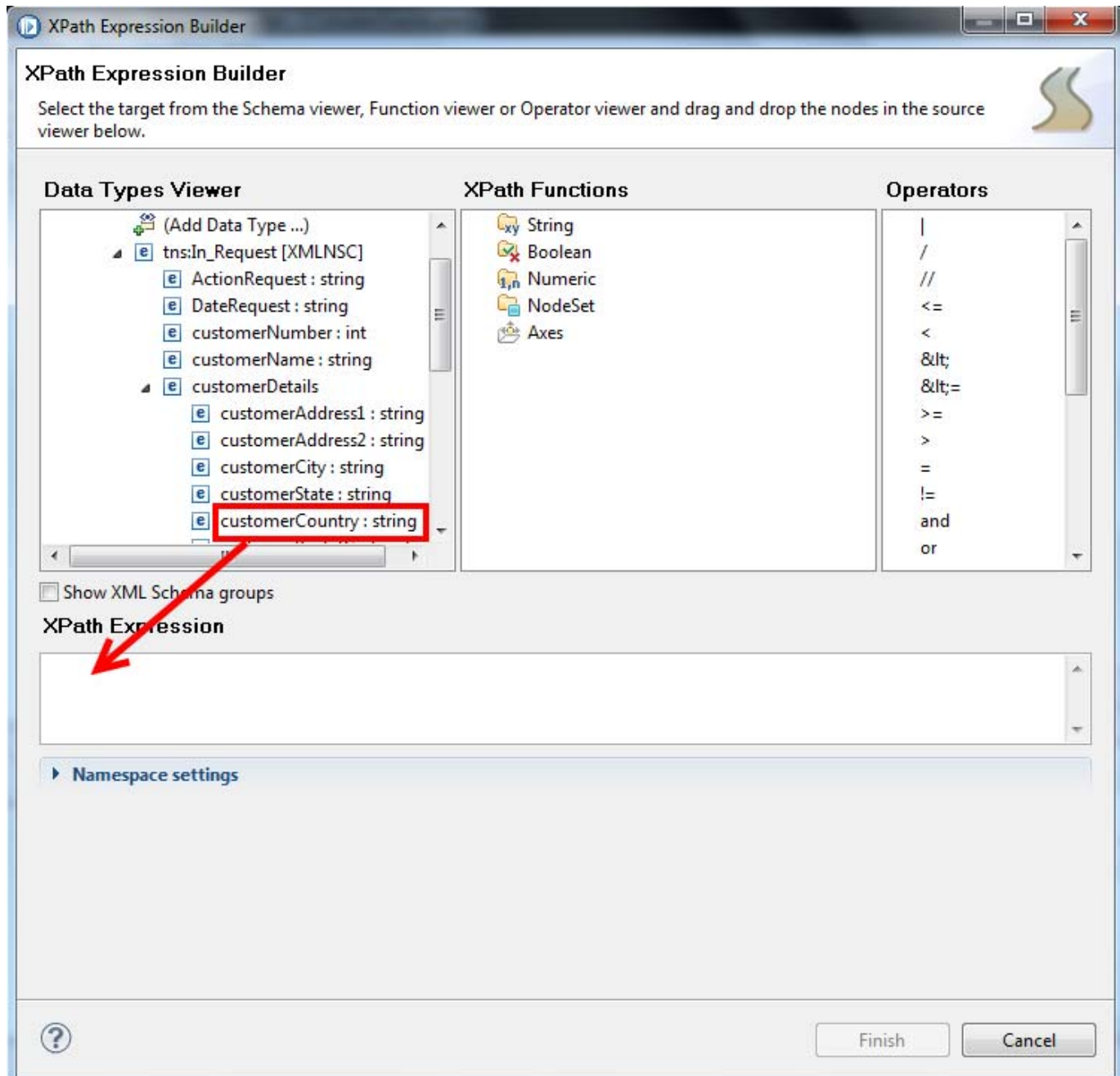
__54. Select **In_Request**.

Note: If you do not see In_Request in the list, then you must add a reference to the Library by right clicking the IntroLab application and selecting **Manage Library references**.

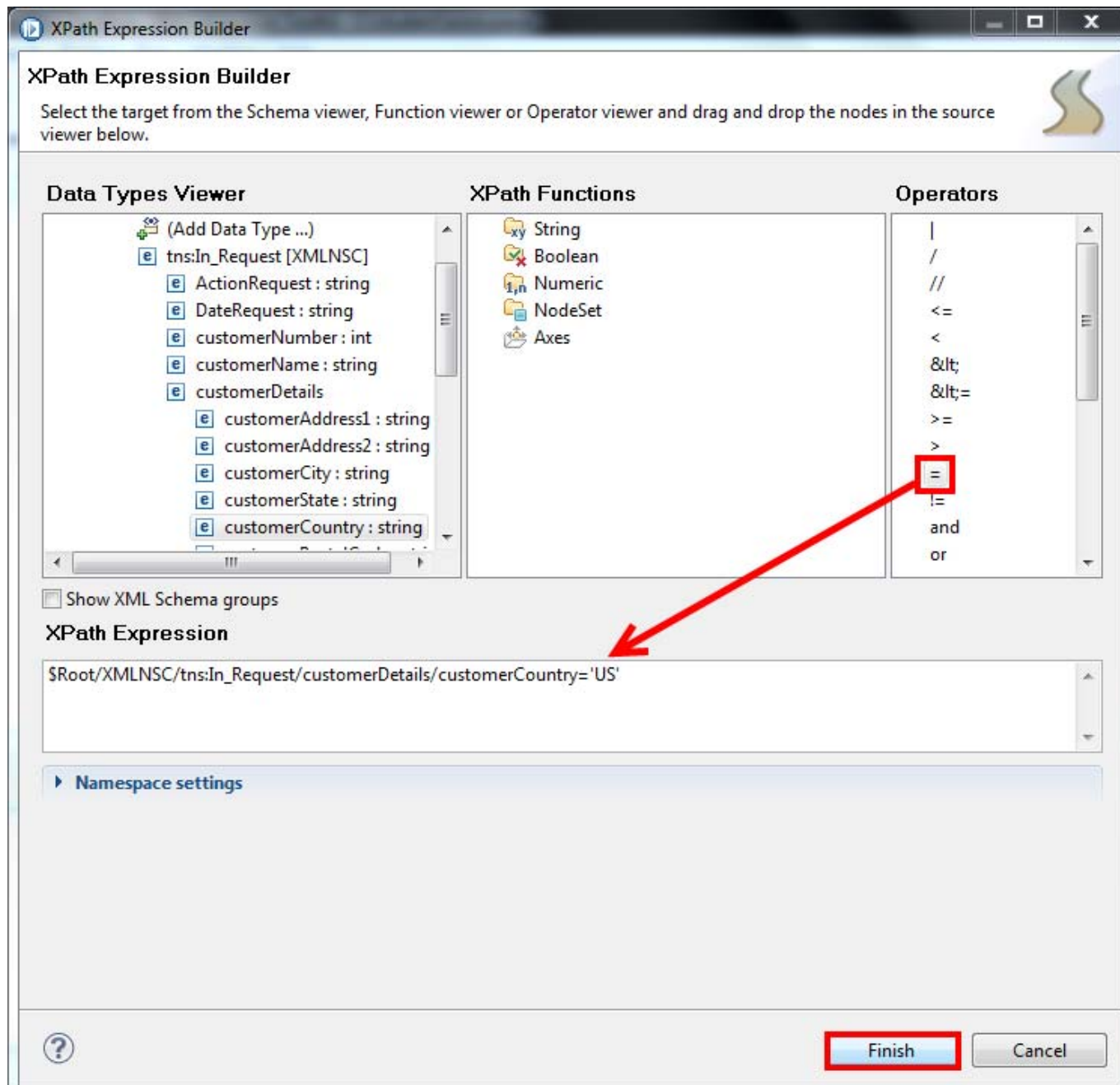
__55. Press the **OK** button to continue.



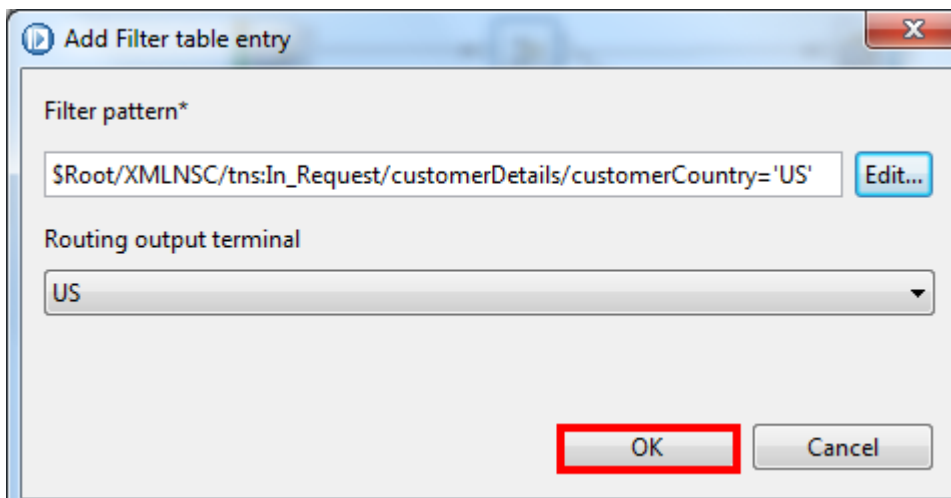
- __56. Expand the **tns:In_Request**→**customerDetails** elements.
- __57. Select the **customerCountry** element and **drag** it into the **XPath Expression** dialog box.



- __58. Drag an equal sign from the **Operators** pane to the end of the expression.
- __59. Append the letters **'US'** (including the single quotes) after the equal sign.
- __60. Press the **Finish** button to complete the XPath expression.

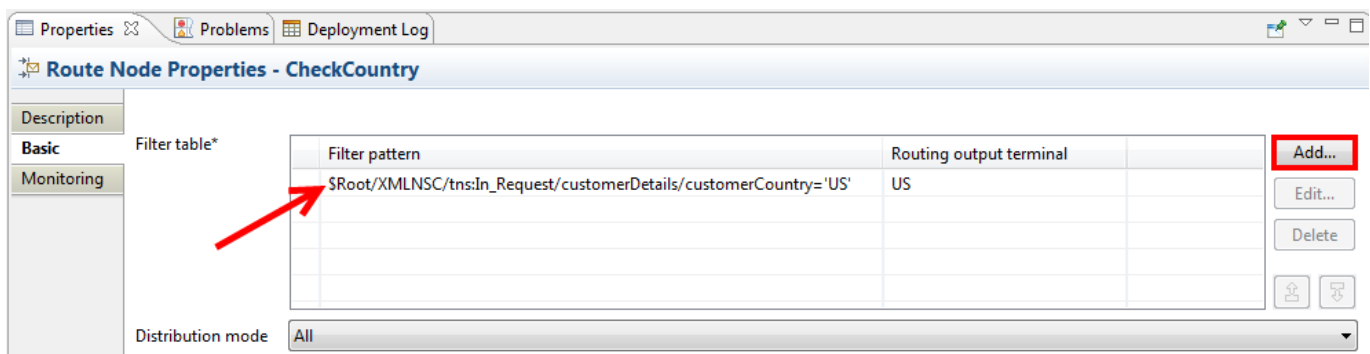


__61. Press the **OK** button to complete the **Filter table entry**.



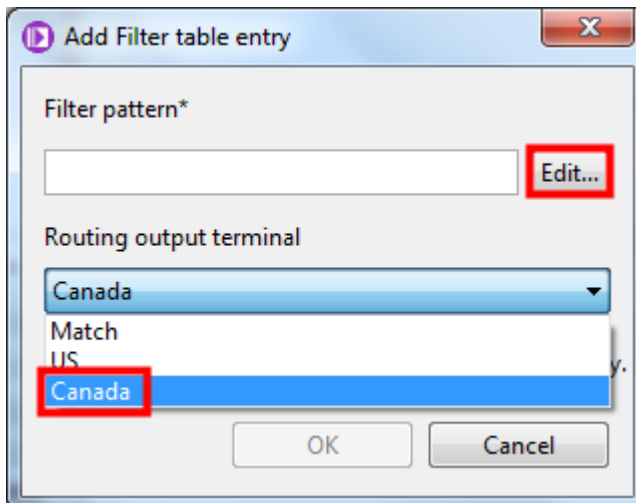
The filter pattern for the **US** terminal should now be visible. The process will now be repeated to create a Filter table entry for the **Canada** terminal.

__62. Press the **Add** button to enter a filter pattern for the **Canada** terminal.

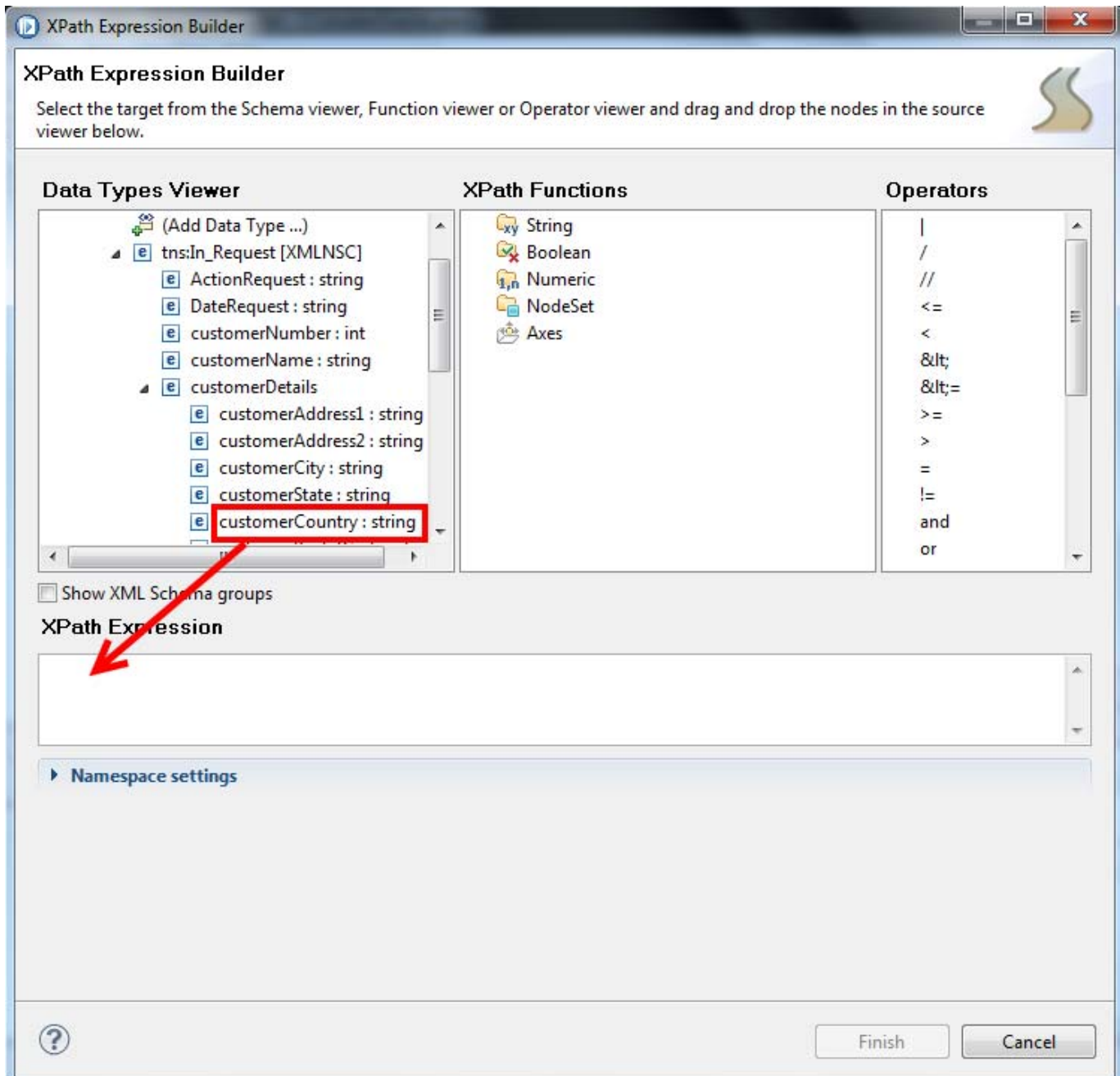


__63. Use the drop-down menu to select the **Canada** terminal as the **Routing output terminal**.

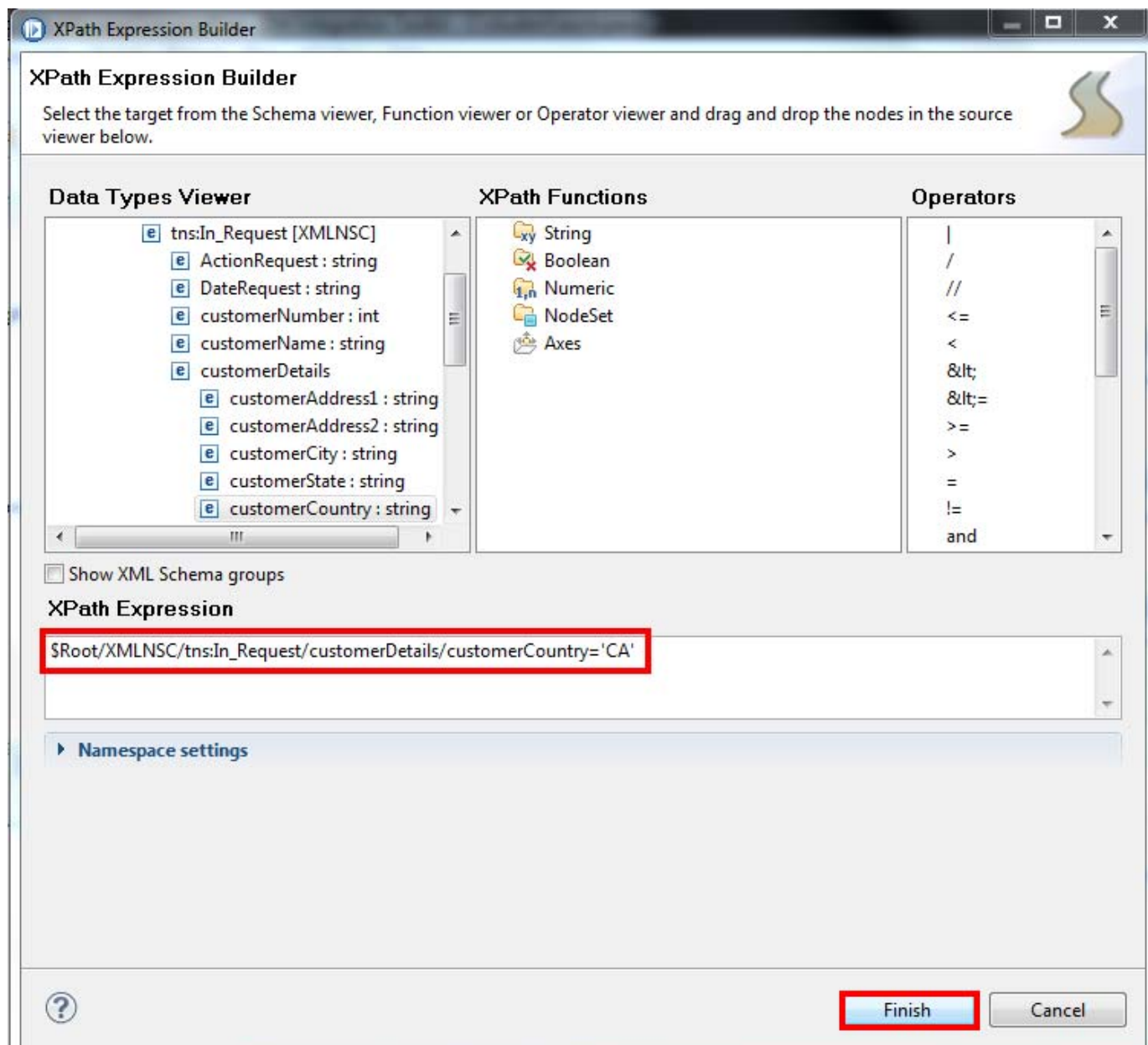
__64. Press the **Edit...** button.



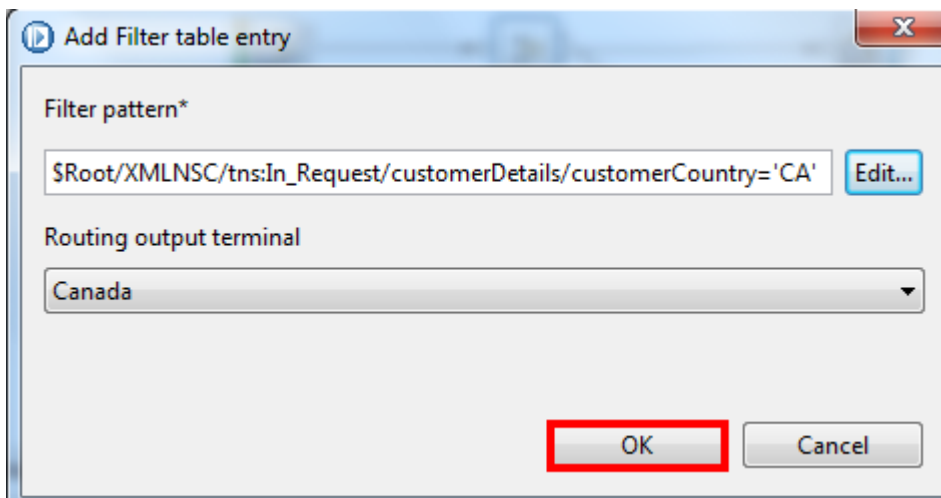
- __65. Expand **Root→tns:In_Request→customerDetails**. Tip: If you do not see In_Request in the Viewer, re-add it as done in steps 53-56.
- __66. Select the **customerCountry** field and drag it to the **XPath Expression** window.



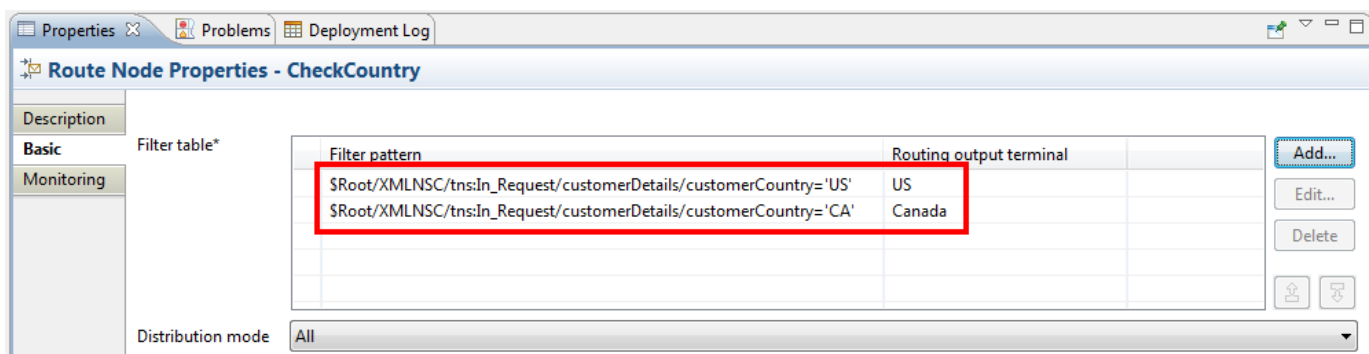
- __67. Complete the XPath Expression by typing `= 'CA'`.
- __68. Press the **Finish** button to complete the XPath expression.



__69. Press the **OK** button to complete the Filter table entry.



The filter pattern for the **Canada** terminal should now be visible.

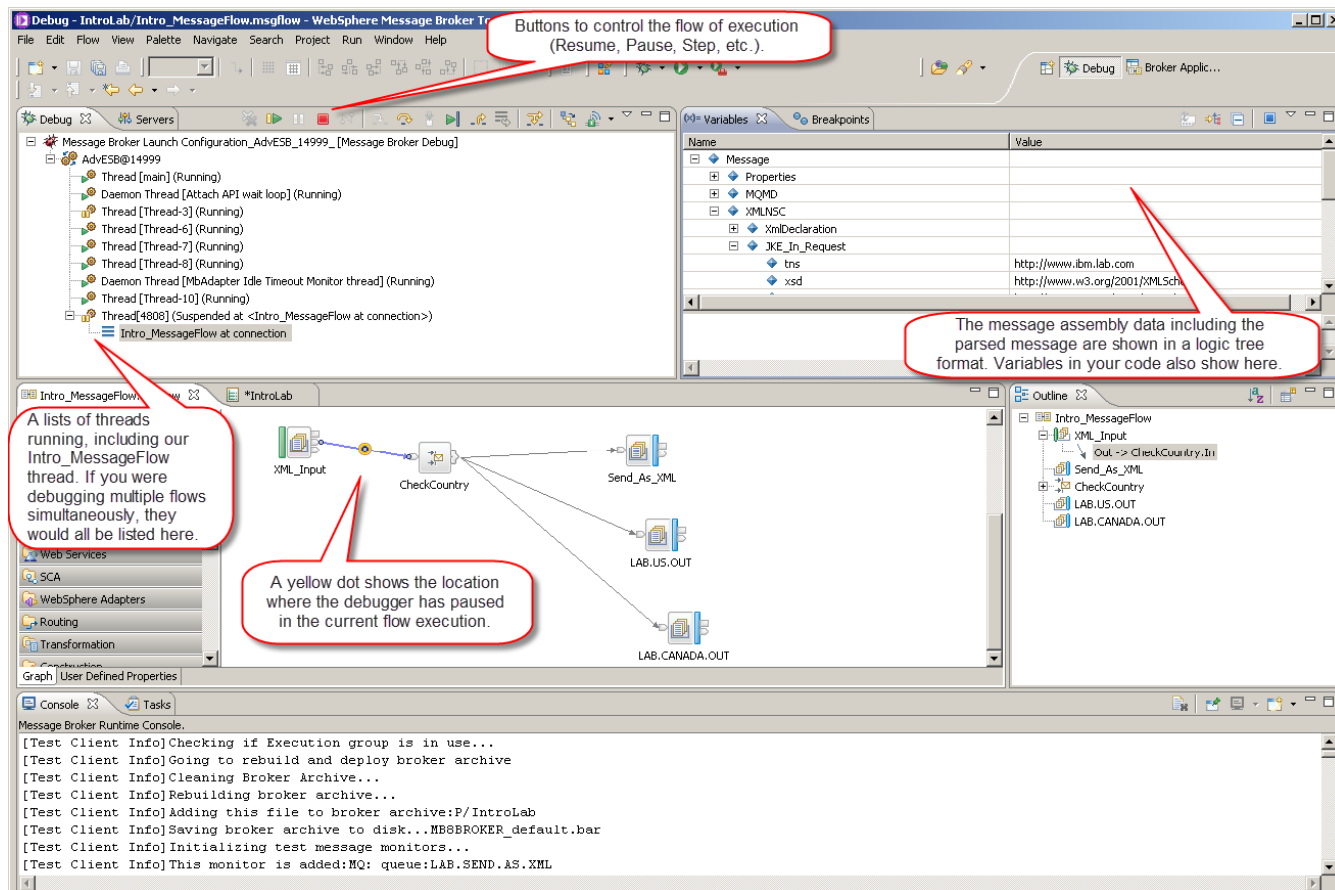


The updates to the message flow are now complete.

__70.  Save the message flow.

3.3 Test with the debugger

Next, we are going to test the application with the graphical debugger.



Key idea: The graphical debugger

Use the flow debugger in the IBM® Integration Bus Toolkit to track messages through your message flows.

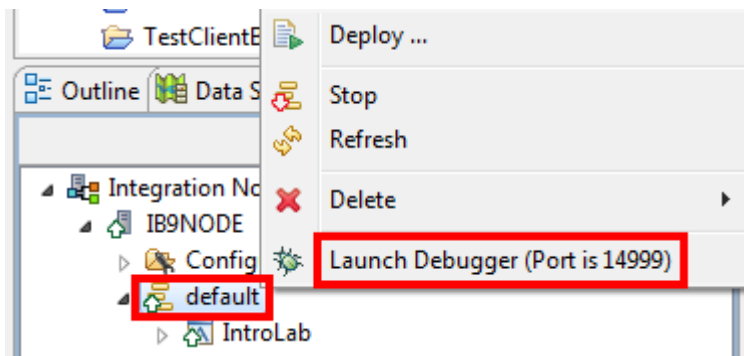
Use the debug perspective in the IBM Integration Toolkit to use the flow debugger. The diagram above introduces the debug perspective and the views it presents.

You can set breakpoints in a message flow, and then step through the flow. While you are stepping through, you can examine and change the message variables and the variables used by ESQL code and Java code. You can debug a wide variety of error conditions in flows, including:

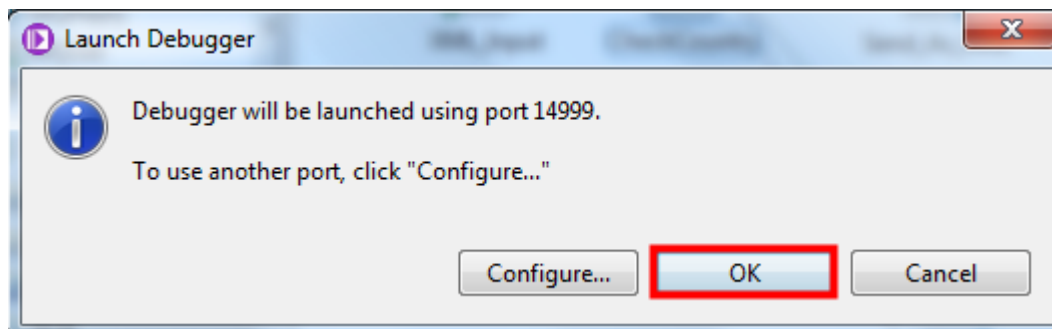
- Nodes that are wired incorrectly (such as outputs that are connected to the wrong inputs)
- Errors in transformation or logic within your code or maps
- Incorrect conditional branching in transition conditions
- Unintended infinite loops in flow

From a single IBM Integration Toolkit, you can attach the debugger to one or more integration servers, and debug multiple flows in different integration servers (and therefore multiple messages) at the same time. However, an integration server can be debugged by only one user at a time. Therefore, if you attach your debugger to an integration server, another user cannot attach a debugger to that same integration server until you have ended your debugging session.

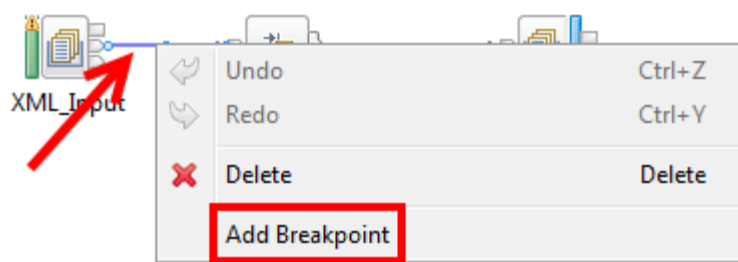
- __1. In the **Integration Node** view in the bottom left, right click the Integration Server called **default**.
- __2. Select **Launch Debugger** from the menu.



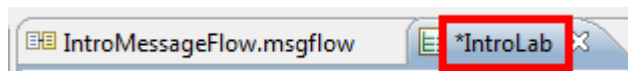
- __3. In the subsequent Launch Debugger dialog, select **OK**.



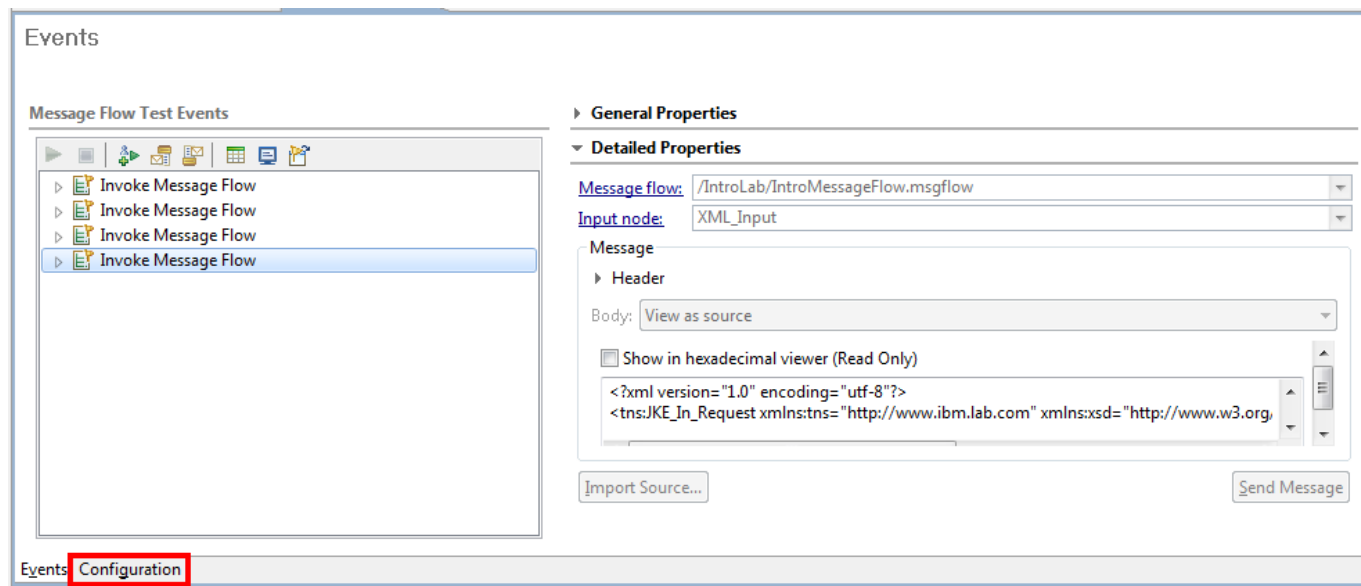
- __4. In the **IntroMessageFlow**, right click the wire between the **XML_Input** node and the **Check Country** route node.
- __5. Select **Add Breakpoint** from the menu.



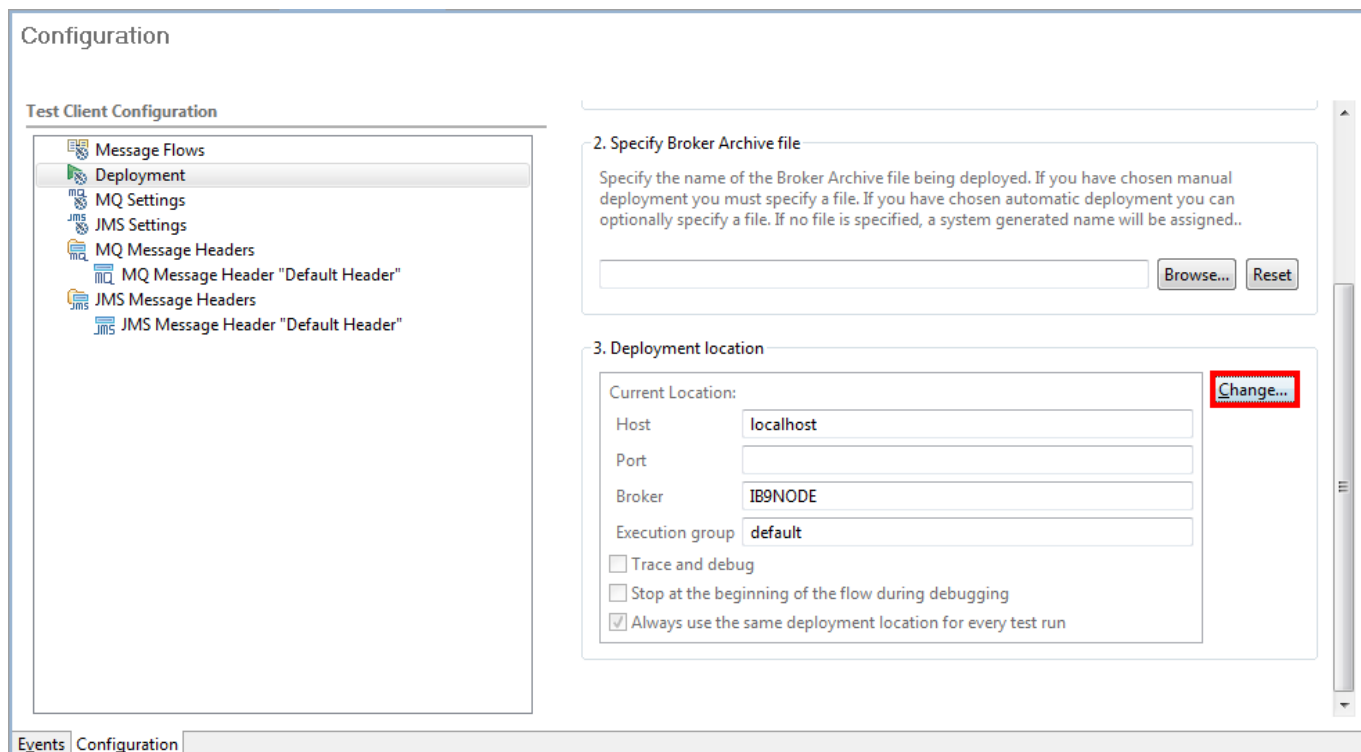
__6. In the main editor view, bring into focus the **IntroLab** Test Client.



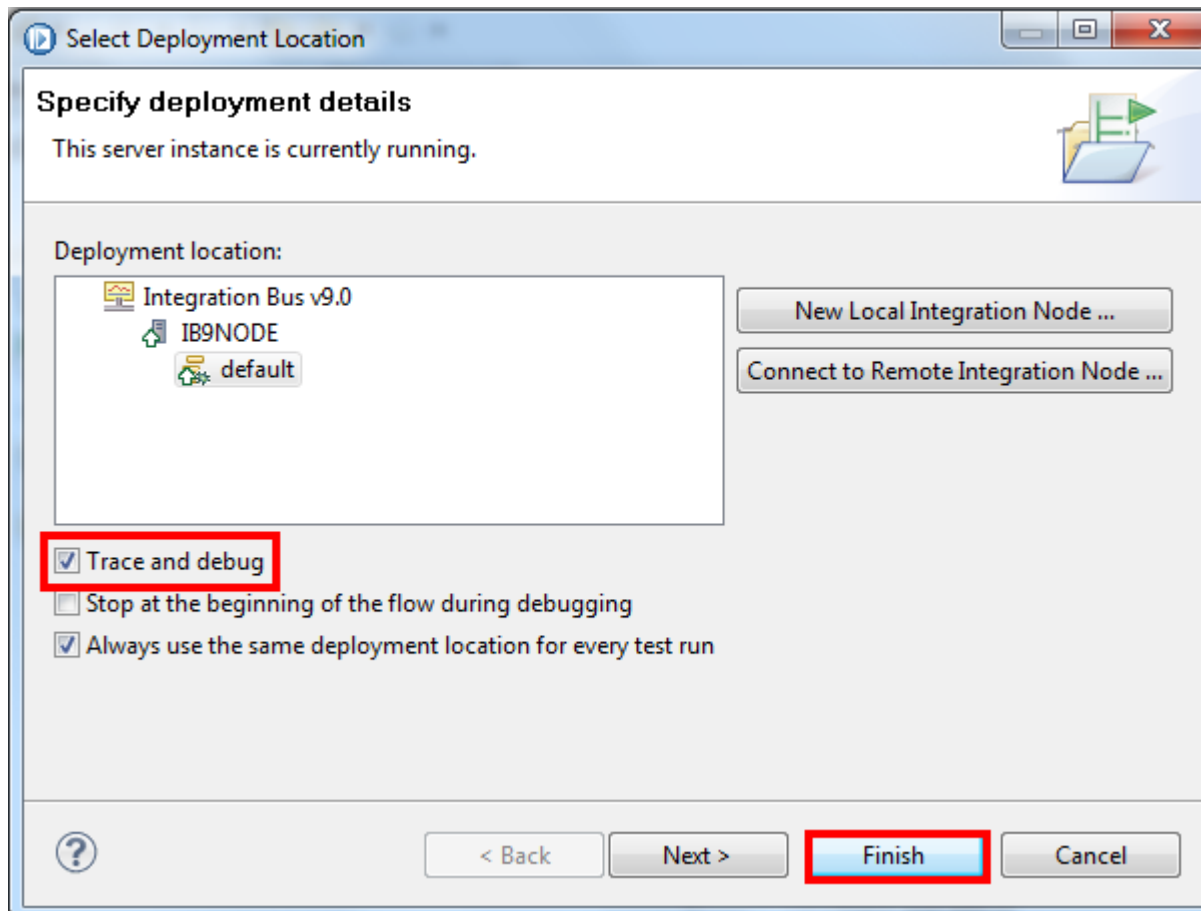
__7. Along the bottom of the Test Client select the **Configuration** tab.



__8. Along the right within the Deployment Location section, click the box called **Change**.

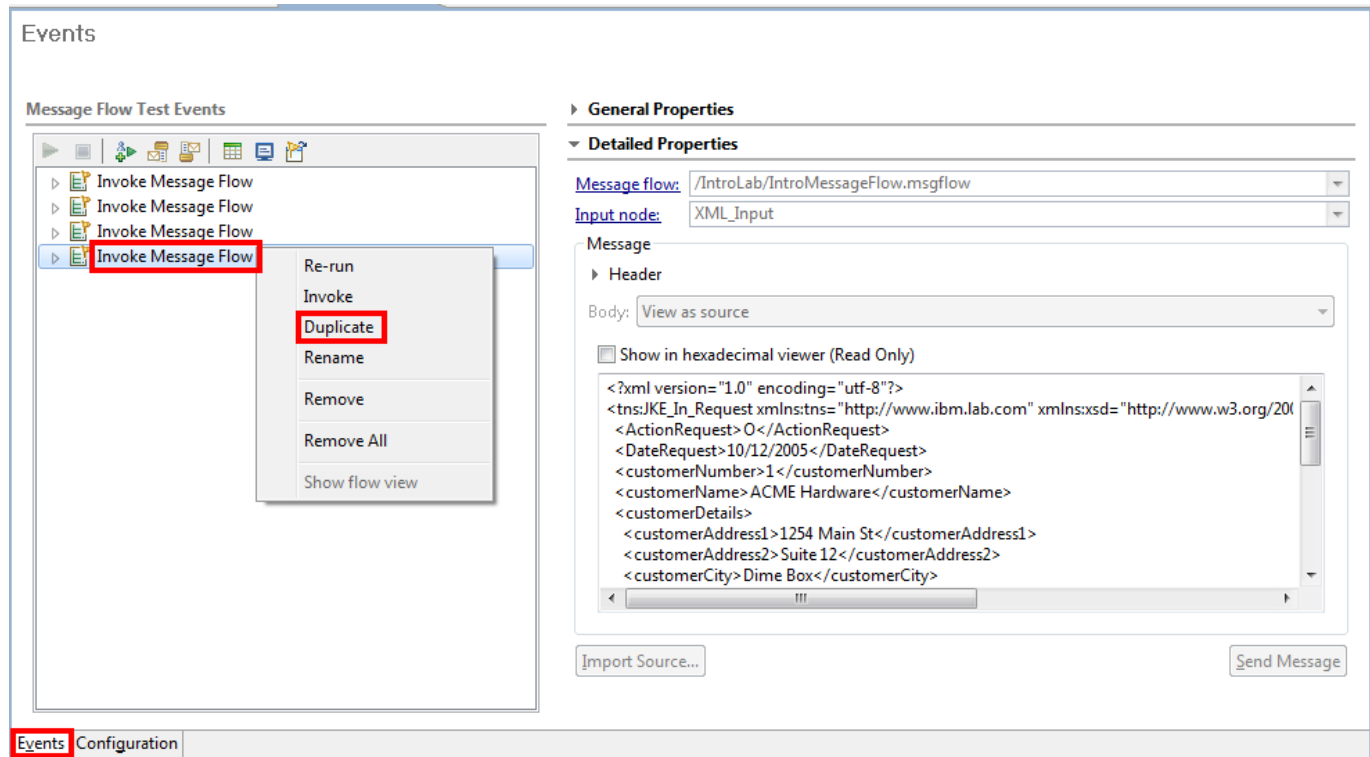


- __9. In the popup dialog, select the checkbox labeled **Trace and Debug**.
- __10. Press the **Finish** button.



The debugger will now run next time the Test Client is run.

- __11. Switch back to the **Events** tab.
- __12. Right click on the last **Invoke Message Flow** in the Message Flow Test Events.
- __13. Select **Duplicate** from the menu.



- ___14. In Detailed Properties, the Body should now show Edit as text. If not, use the drop-down to select it.

The screenshot displays the 'Events' tab in the IBM Software interface. On the left, the 'Message Flow Test Events' pane shows a list of 'Invoke Message Flow' events. The right pane, titled 'Detailed Properties', shows the configuration for the selected message flow. The 'Message flow' is set to '/IntroLab/IntroMessageFlow.msgflow' and the 'Input node' is 'XML_Input'. The 'Message' section is expanded, showing the 'Body' dropdown set to 'Edit as text'. Below this, the XML body content is displayed in a text area, with the value 'USA' highlighted. At the bottom of the right pane, there are buttons for 'Import Source...' and 'Send Message'.

Events

Select the message flow you would like to test. Click Send Message to run.

Message Flow Test Events

General Properties

Detailed Properties

Message flow: /IntroLab/IntroMessageFlow.msgflow

Input node: XML_Input

Message

Header

Body: Edit as text

```
<ActionRequest>0</ActionRequest>
<DateRequest>10/12/2005</DateRequest>
<customerNumber>1</customerNumber>
<customerName>ACME Hardware</customerName>
<customerDetails>
  <customerAddress1>1254 Main St</customerAddress1>
  <customerAddress2>Suite 12</customerAddress2>
  <customerCity>Dime Box</customerCity>
  <customerState>TX</customerState>
  <customerCountry>USA</customerCountry>
</customerDetails>
</Message>
```

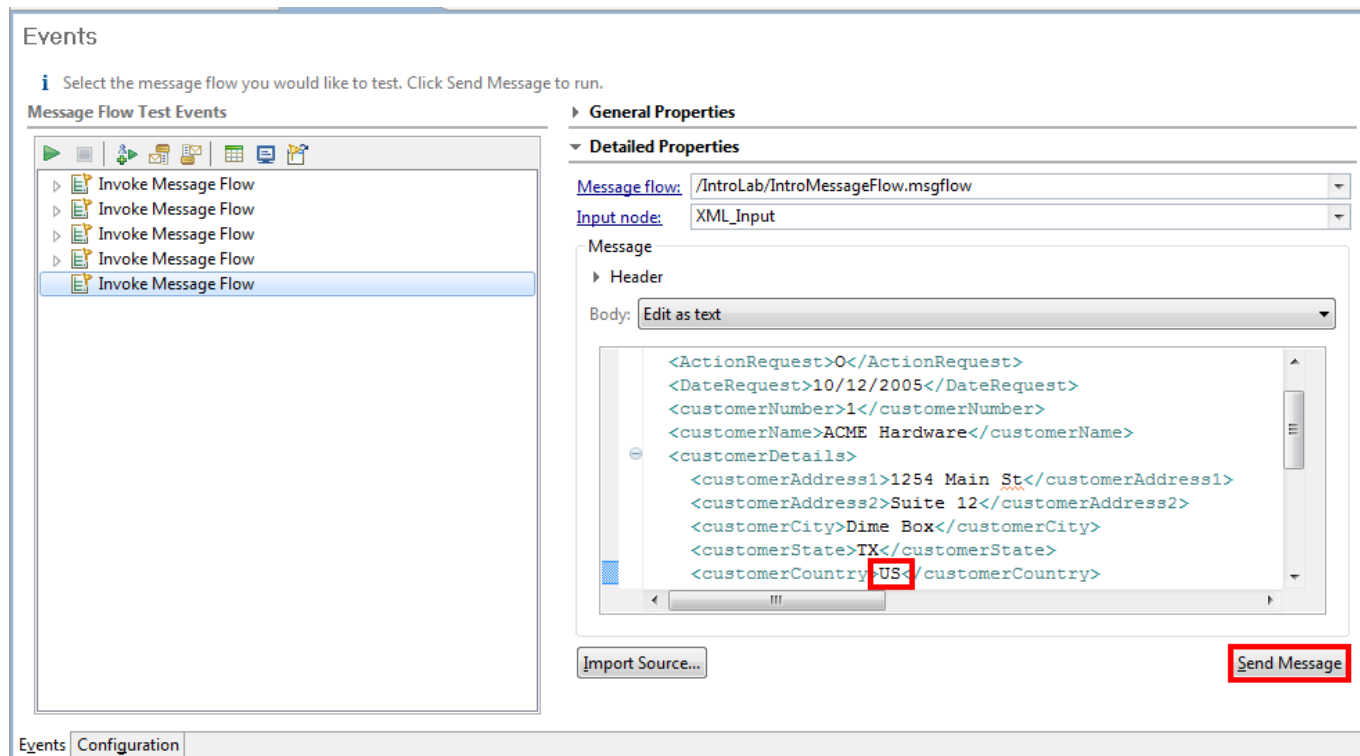
Import Source... Send Message

Events Configuration

__15. Edit the input message. Locate the customerCountry element.

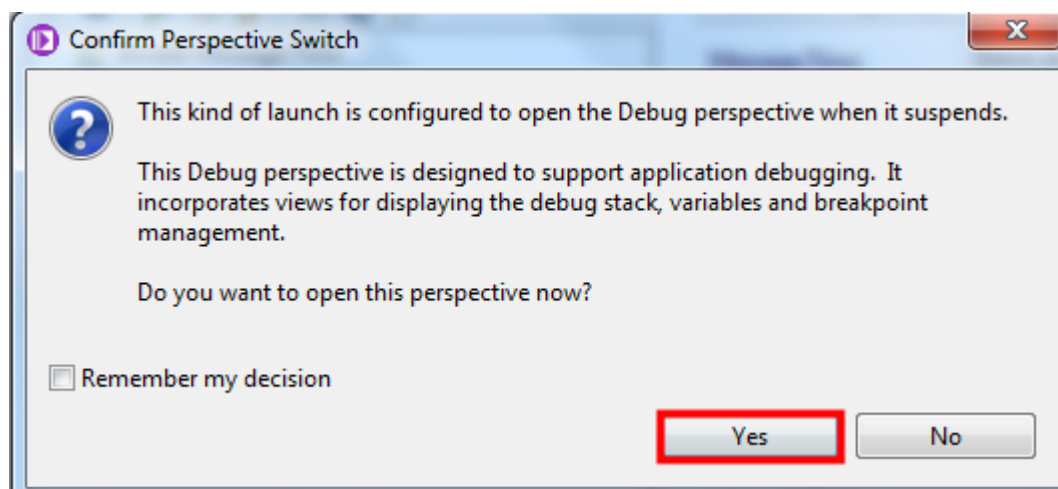
__16. Change the data value to **US**.

__17. Run the test client by clicking **Send Message**.

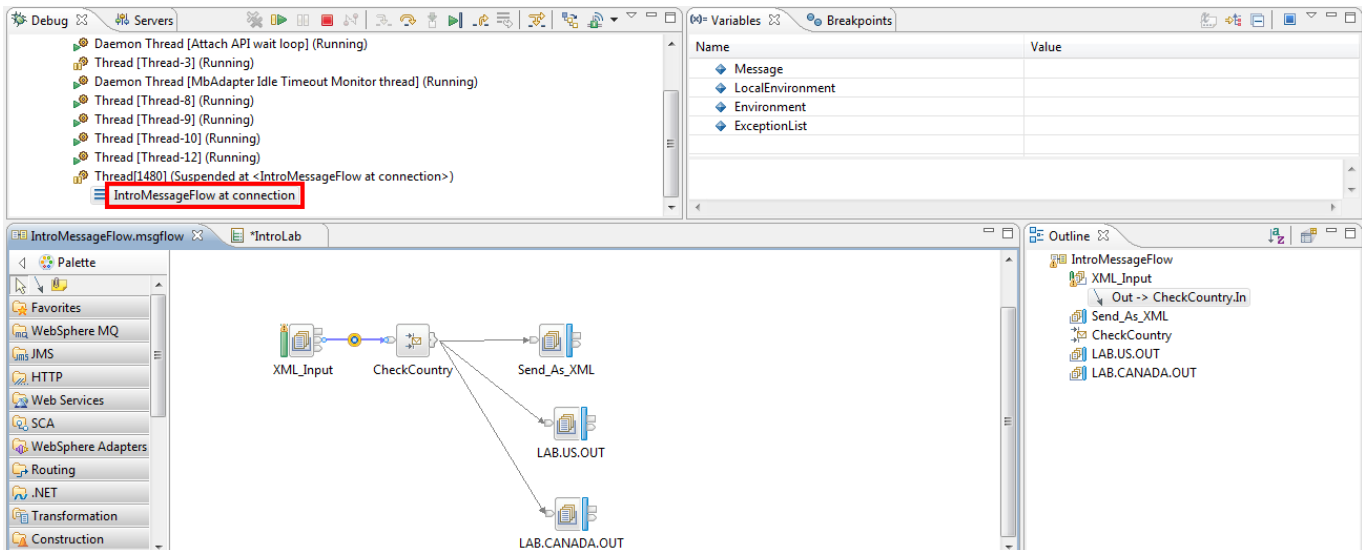


A popup will confirm switching the Eclipse perspective to the Debug perspective.

__18. Press **Yes**.

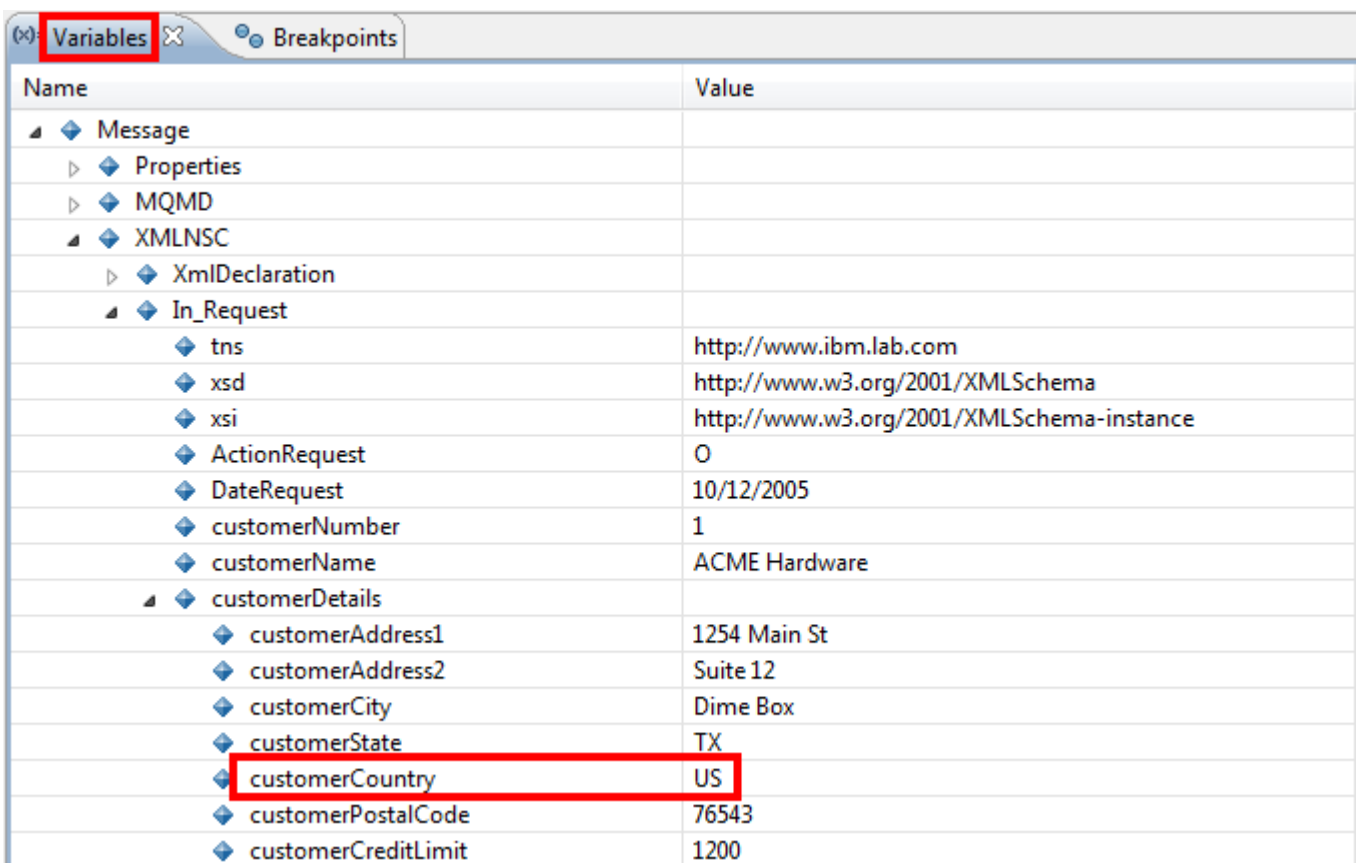


- __19. Ensure that the **IntroMessageFlow** thread is selected in the top left view. You should see a yellow halo around the breakpoint. The yellow halo indicates that this is where execution of the flow has been suspended.

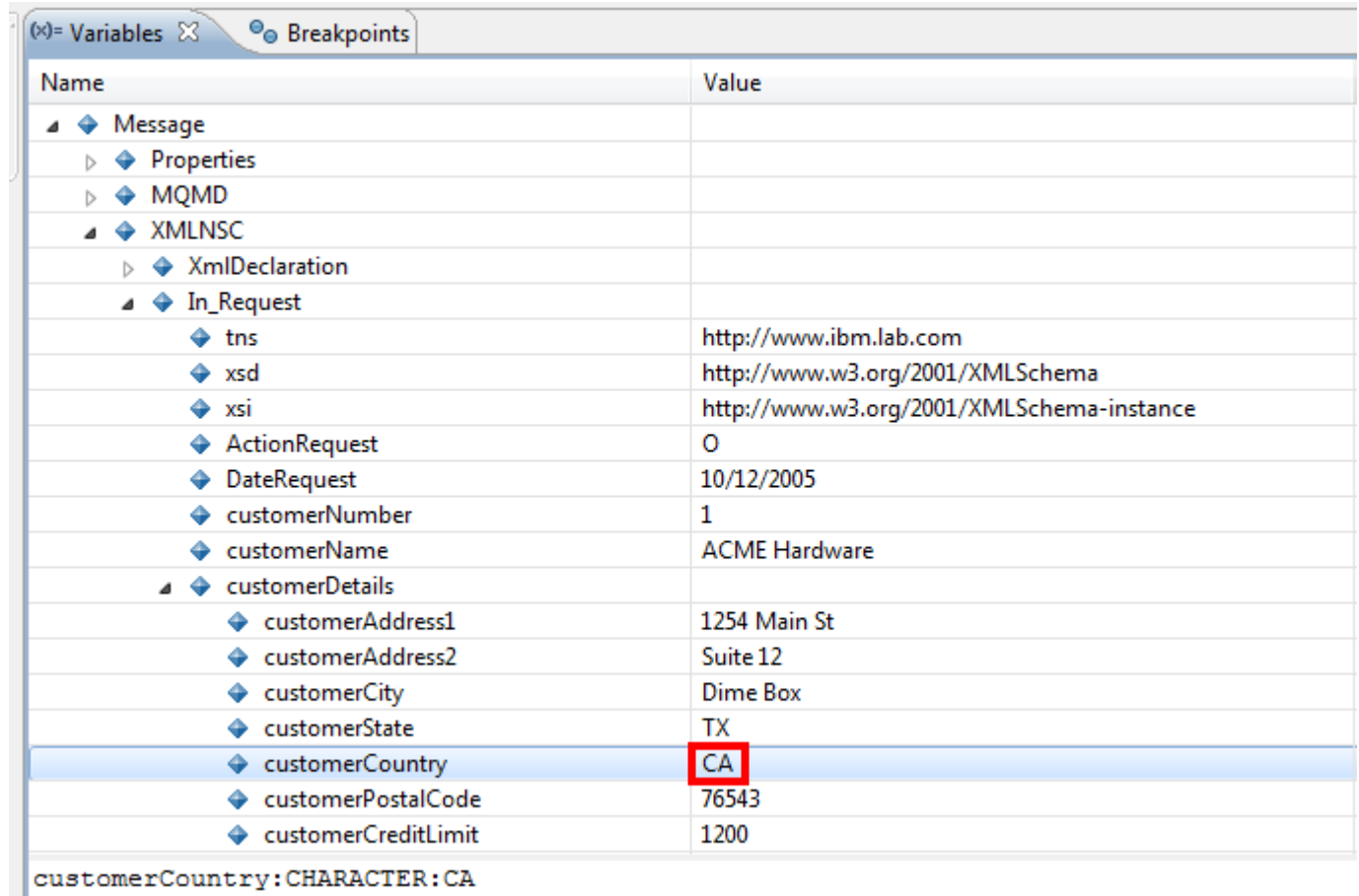


- __20. In the top right Variables view, expand **Message→XMLNSC→In_Request→customerDetails**.

- __21. You will see that the value is set to **US**.



- __22. To interactively change this, single click the **Value** column of the **customerCountry** element.
- __23. Change the value in the editor box to **CA**.
- __24. Press **Enter** to update the value.



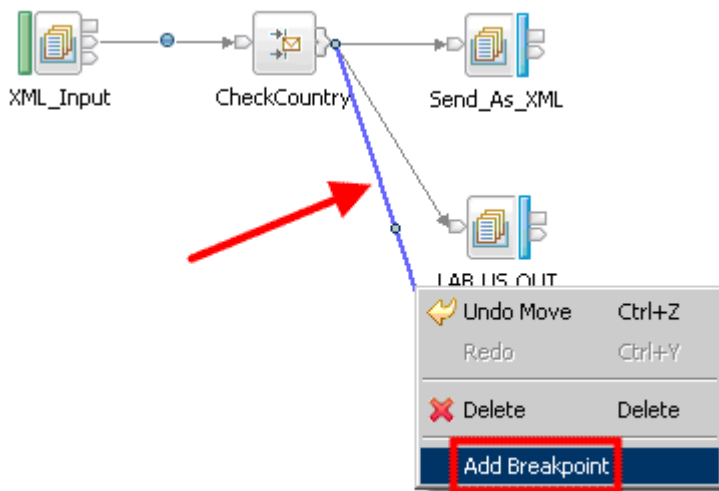
The screenshot shows the 'Variables' window in IBM Integration Bus. The 'In_Request' node is expanded, showing a tree of variables. The 'customerCountry' variable is selected, and its value 'CA' is highlighted in the 'Value' column. A red box is drawn around the 'CA' value. Below the table, the text 'customerCountry: CHARACTER: CA' is displayed.

Name	Value
Message	
Properties	
MQMD	
XMLNSC	
XmlDeclaration	
In_Request	
tns	http://www.ibm.lab.com
xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
ActionRequest	0
DateRequest	10/12/2005
customerNumber	1
customerName	ACME Hardware
customerDetails	
customerAddress1	1254 Main St
customerAddress2	Suite 12
customerCity	Dime Box
customerState	TX
customerCountry	CA
customerPostalCode	76543
customerCreditLimit	1200

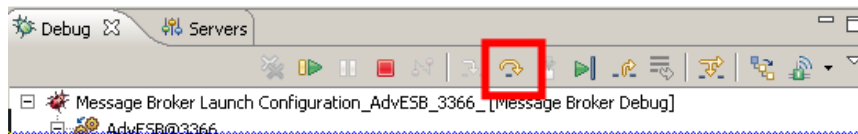
customerCountry: CHARACTER: CA

__25. Right click the wire between the **CheckCountry** node and the **LAB.CANADA.OUT** node.

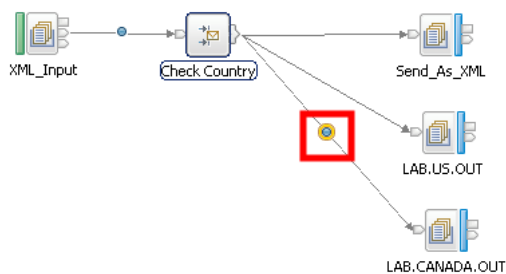
__26. Select **Add Breakpoint** from the menu.



__27. Select the **Step Over** button along the actions menu panel to run the Route node logic. Alternatively, you can press F6 to step over.



__28. Verify that the Route node has sent the message down the Canada terminal.



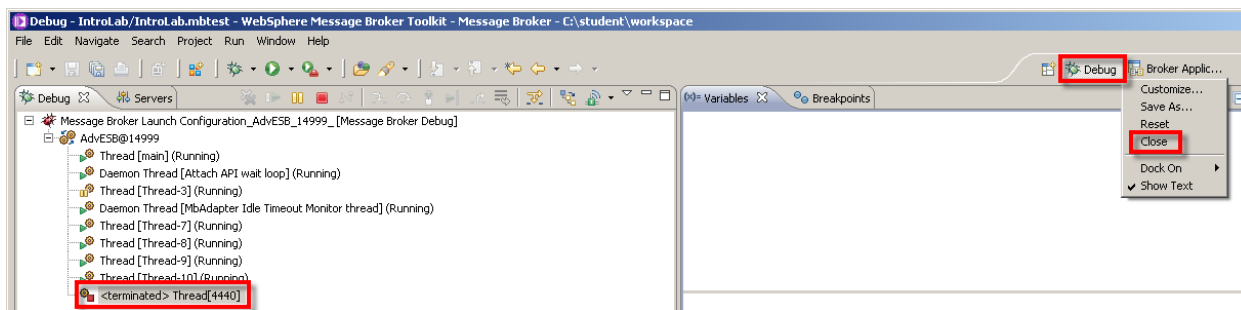
__29. Select Resume or Step Over to finish the flow (or press F8 or F6, respectively).



You can now re-run the test without overriding the customerCountry element in order to validate the path to the **LAB.US.OUT** queue. Return to the Integration Development perspective and the Test Client, and “Re-run” the last Invoke Message Flow.

___30. When finished testing the debugger (the debugger should be in a terminated state), right click the **Debug** perspective.

___31. Select **Close** from the menu.



3.4 A closer look at the deployment process

Up to this point, we have been using the Test Client to initiate our unit testing and it has been handling the deployment process for us “under the covers.” To finish this lab, we will briefly examine the deployment process and manually do our own deployment.

Key idea: The deployment process

When you create application resources such as message flows in the IBM Integration Toolkit, you must distribute them to the nodes on which you want them to run. Data for message flows and associated resources is packaged in a broker archive, or BAR, file before being sent to the node.

You can initiate a deployment in the following ways:

- From the IBM Integration Toolkit
- From the Integration Explorer
- By using the **mqsidedeploy** command
- By using functions defined by the Integration Bus CMP API

Depending on your work patterns, you might use all these methods at different times.

The Integration Toolkit provides a Nodes view in the lower left-hand corner of the Integration Development perspective. If you expand an integration node, all the integration servers in that node are displayed, as well as deployed message flows and their associated resources. You can drag an Application or Library, message flow, or a BAR file from the Application Development view onto an execution group to deploy it. Alternatively, you can right-click on an execution group to select an Application or Library, message flow, or BAR file to deploy to the selected execution group.

If you are working with an application and want to deploy and test it quickly, you can deploy just that resource. Drag the resource onto the execution group to which you want to deploy it. A BAR file is generated automatically, and deployed to the node. If libraries are referenced, they are added automatically to the BAR file and deployed. If a message flow contains a subflow that is defined in a “.subflow” file, the subflow is automatically included in the BAR file, and deployed with the message flow. If you drag a flow that is contained in an Application or Library, you will see a message saying that the whole application or library will be deployed, because you cannot deploy a message flow on its own if it belongs to an Application or Library.

Key idea: The BAR file

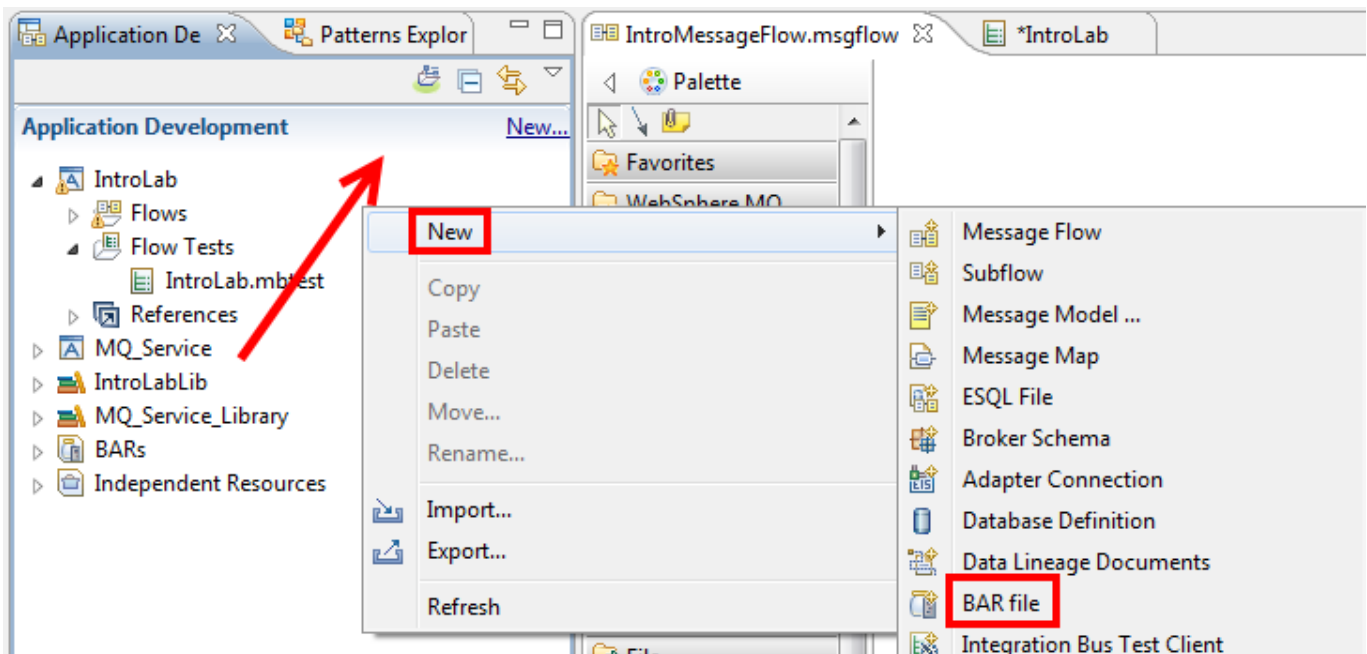
The unit of deployment to a broker is a BAR file. It is a .zip file that contains the flows, models, .jar files, maps, and any other resources in the workspace needed to run applications. The BAR file also contains a deployment descriptor .xml file, which exposes flow and node properties for override at build or deploy time. The following sequence of events illustrates how to deploy with a BAR file:

1. Create a broker archive.
2. Add files to the broker archive.
3. If necessary, edit the configurable properties of the message flows or applications in the broker archive.
4. Deploy the BAR file by sending it to the broker, from where its contents are distributed to the integration servers.

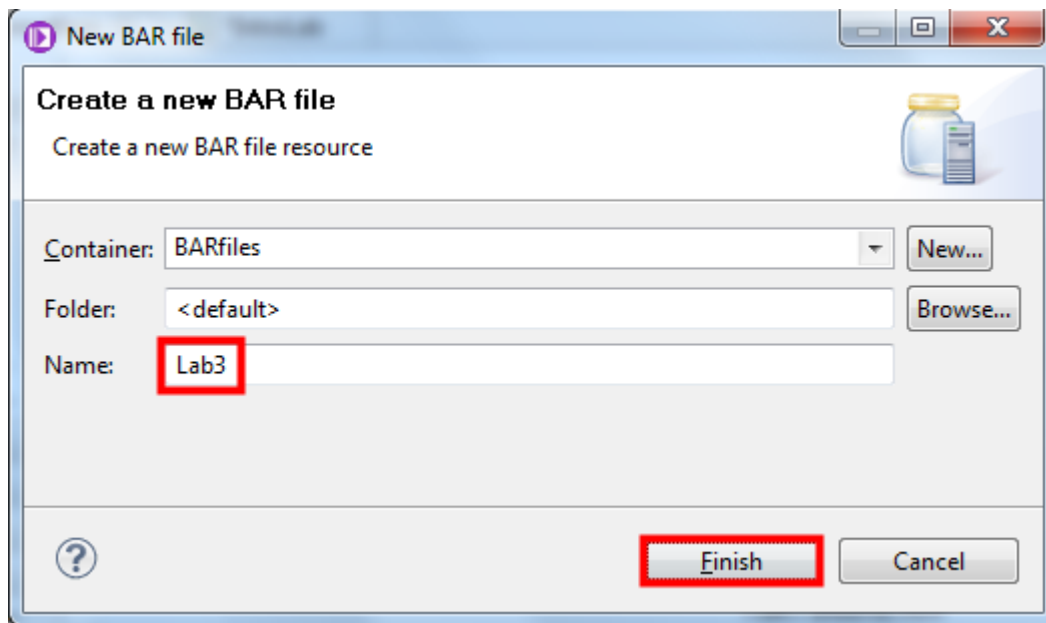
A BAR file can be deployed in two ways:

- [Incremental BAR file deployment](#). Deployed files are added to the execution group. Files that exist in the execution group are replaced by the new version.
- [Complete BAR file deployment](#). Files that are already deployed to the execution group are removed before the entire contents of the BAR file are deployed. Therefore, nothing is left in the execution group from previous deployments.

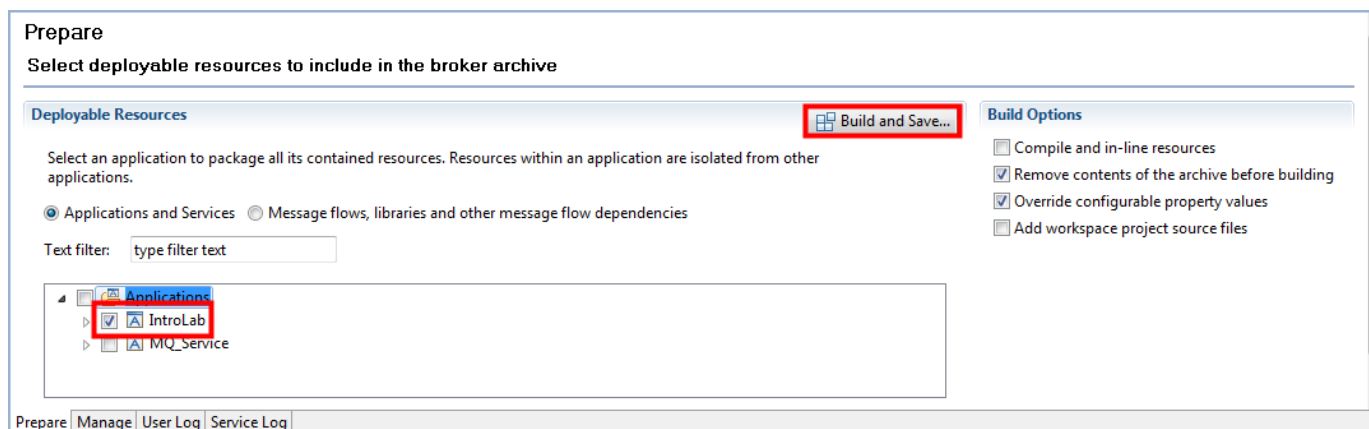
- __1. Return to the Integration Development perspective in the toolkit.
- __2. Select a blank area in the **Navigation** view on the left.
- __3. Press the right mouse button.
- __4. Select **New**→**BAR** file from the menu.



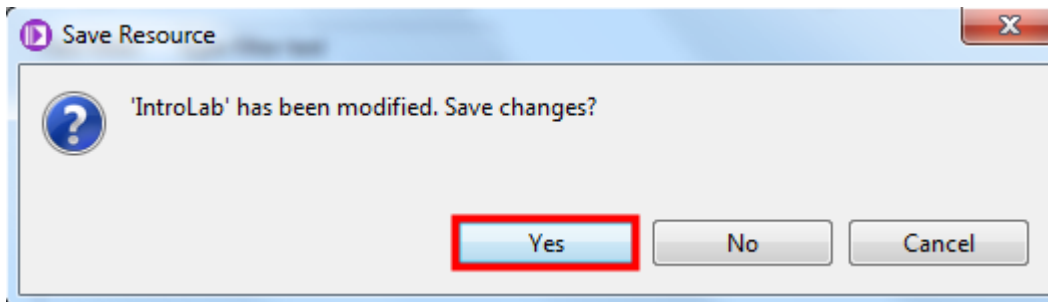
- __5. Enter **Lab3** as the name of the new broker archive file.
- __6. Select **Finish**.



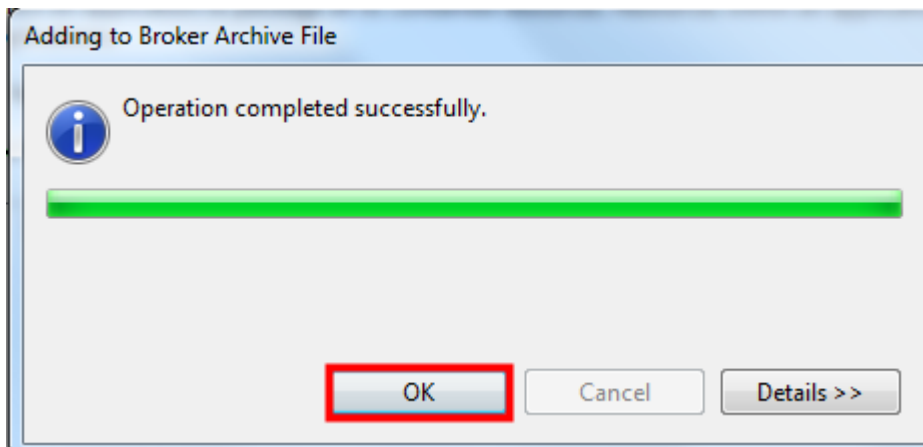
- __7. In the BAR editor, select the **IntroLab** Application.
- __8. Click the **Build and Save...** button.



__9. If prompted to Save, click **Yes**.



__10. Press the **OK** button.



- ___11. In the BAR editor, select the **Manage** tab.
- ___12. Expand the **IntroLab** app and select the various resources.
- ___13. Look at the **Properties view** below to see what properties are exposed in order to be overridden within the BAR file. For example, select the **LAB.US.OUT** node. You can see that the Queue Name property can be overridden at deployment time.

Manage
Rebuild, remove, edit, add resources to broker archive and configure their properties

Filter by: <Type filter text>

Name	Type	Modified	Size	Path	Vers...	Comment
IntroLab	Application	Jul 10, 2013 2:50:12 PM	4799			
IntroLabLib	Library	Jul 10, 2013 2:50:12 PM	1749			
XML Schemas and WSDL						
IN_Request.xsd	XSD file	Jul 10, 2013 2:50:12 PM	2070			
IntroMessageFlow.msgflow	Message flow	Jul 10, 2013 2:50:12 PM	3983		1.0	
IntroMessageFlow						
CheckCountry						
LAB.CANADA.OUT						
LAB.US.OUT						
Send_As_XML						
XML_Input						

Command for packaging the BAR contents

Prepare | **Manage** | User Log | Service Log

Properties | Problems | Deployment Log | Progress View

LAB.US.OUT

Configure
Workload Management

Configure properties of selected built resource.

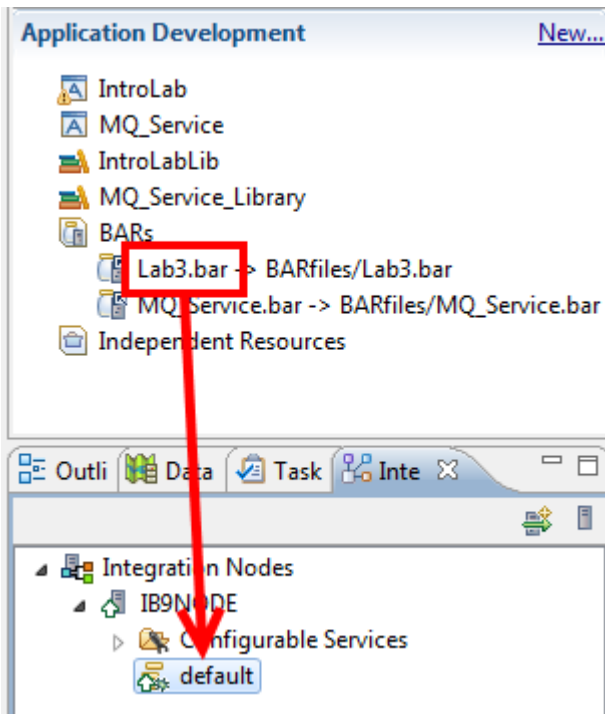
Queue manager name

Queue name **LAB.US.OUT**

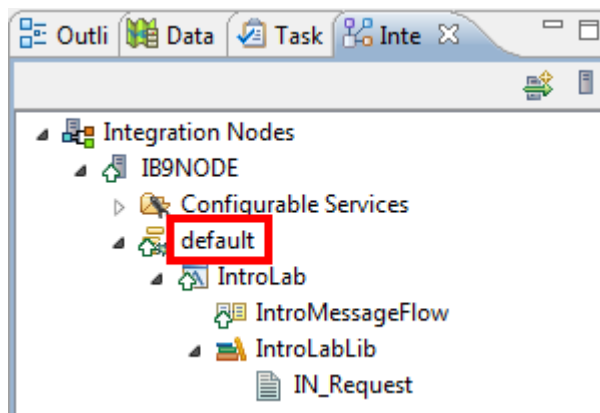
Reply-to queue

__14. To deploy, find the **Lab3.bar** file in the navigator in the **BARs** container.

__15. Drag it onto the **default** integration server of the **IB9NODE**.



__16. Once deployment is complete, click the **default** integration server to open it and see the assets deployed to it. The IntroLab application has been deployed, which included the Intro_MessageFlow message flow and the library that was included, which contains the IN_Request XML Schema.

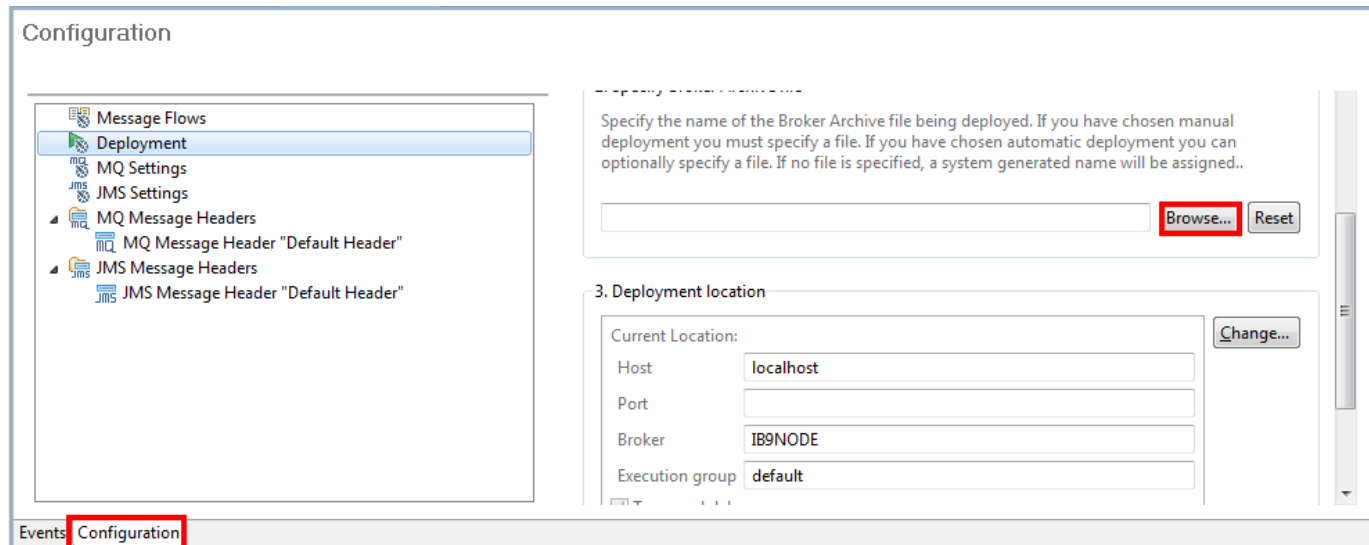


Finally, the Test Client will be updated to use this BAR rather than generating its own.

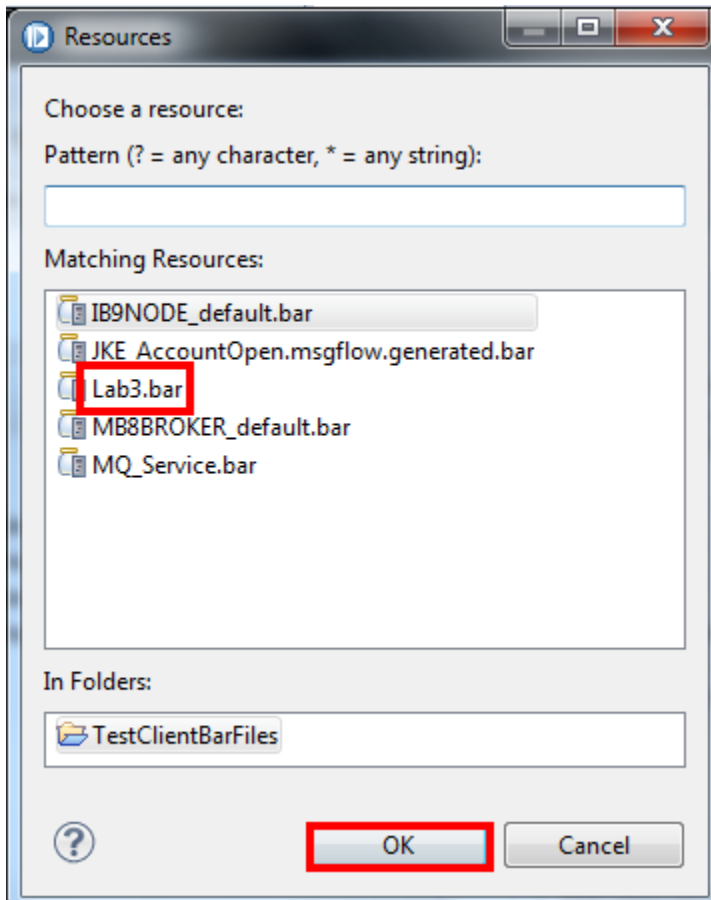
__17. In the editor, select the **Test Client**.



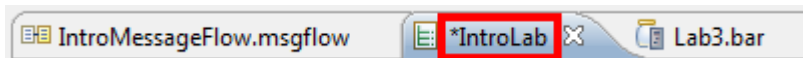
- ___18. Select the **Configuration** tab.
- ___19. Select the **Deployment** tab.
- ___20. On the right inside the **Specify Broker Archive** box, select **Browse**.



- __21. In the dialog box, select **Lab3.bar**.
- __22. Click **OK**.



- __23.  Save the Test Client.



- __24. Close all the open editor tabs but leave the toolkit running.

This is the end of Lab 3.

Lab 4 Building a web service using patterns and mapping

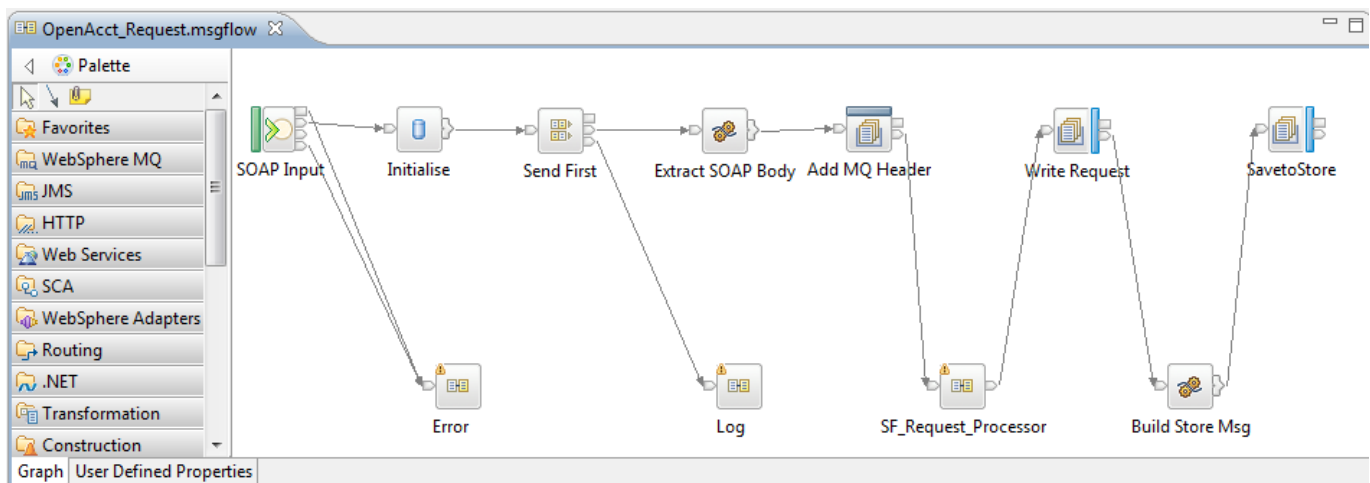
4.1 Lab overview

In this lab, a web service façade will be built on top of an existing IBM WebSphere MQ-based service.

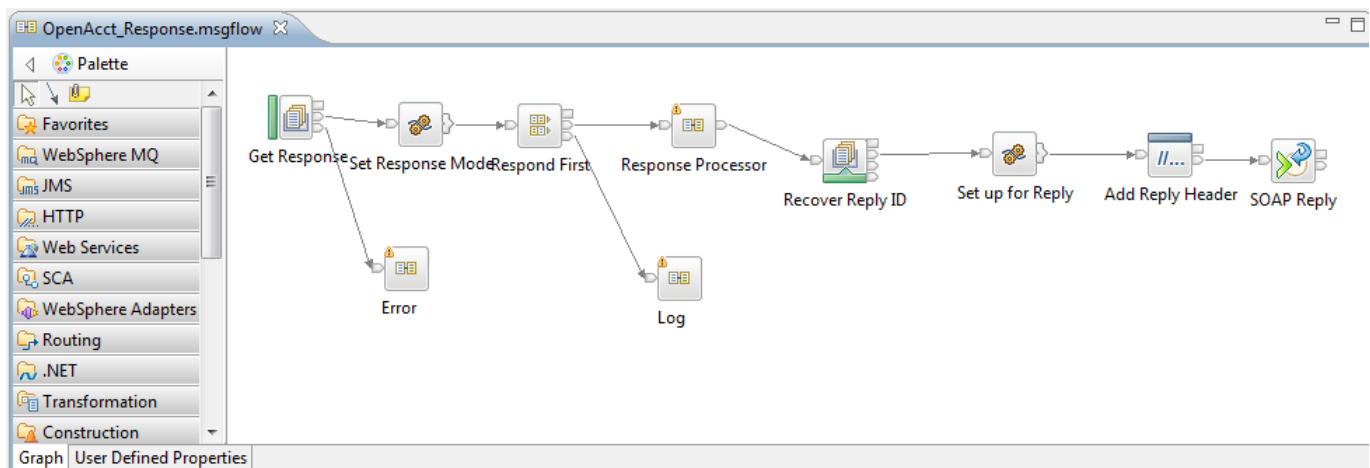
The Patterns Wizard will be used to build a message set and a set of message flows based on a Web Services Description Language (WSDL) definition file that is provided. Using a pattern will produce a solution that includes additional function but that requires less effort to create.

The generated pattern will be extended by defining the mapping between the back-end service format (based on COBOL copy books) and the web service data format (based on the imported WSDL file). The message flow will be tested using the Test Client.

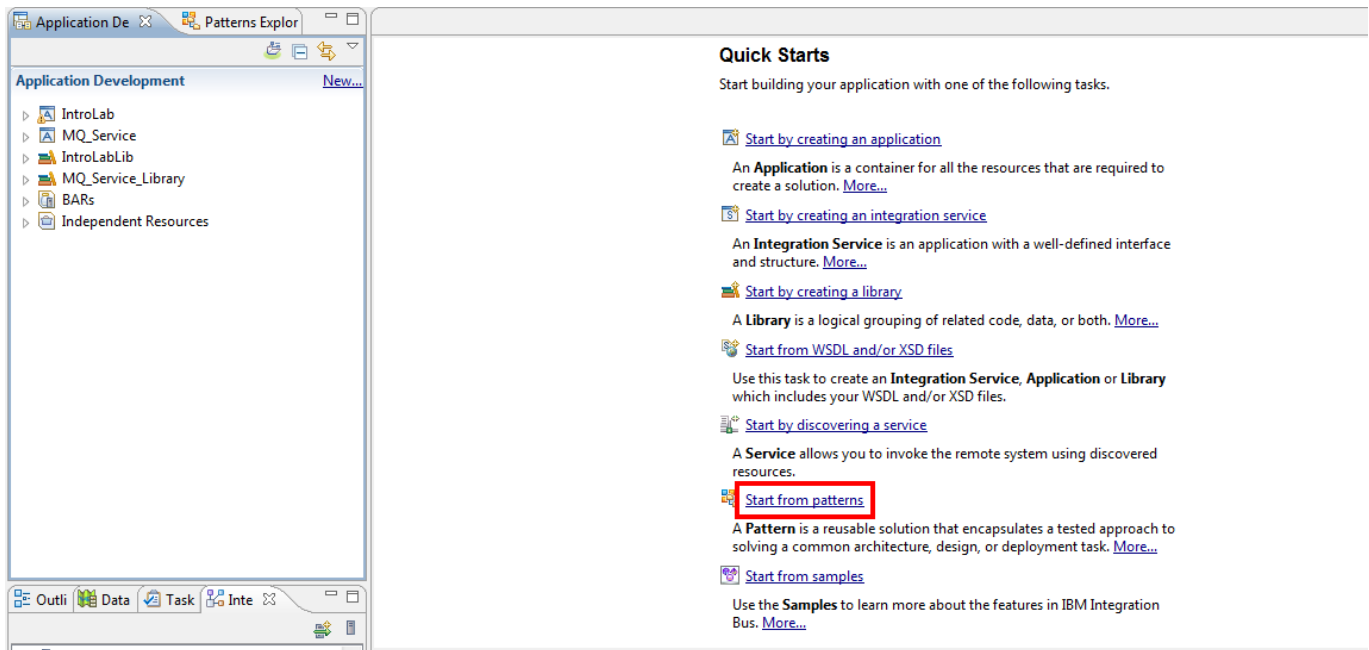
The following is what will be built and tested. This is the main request flow.



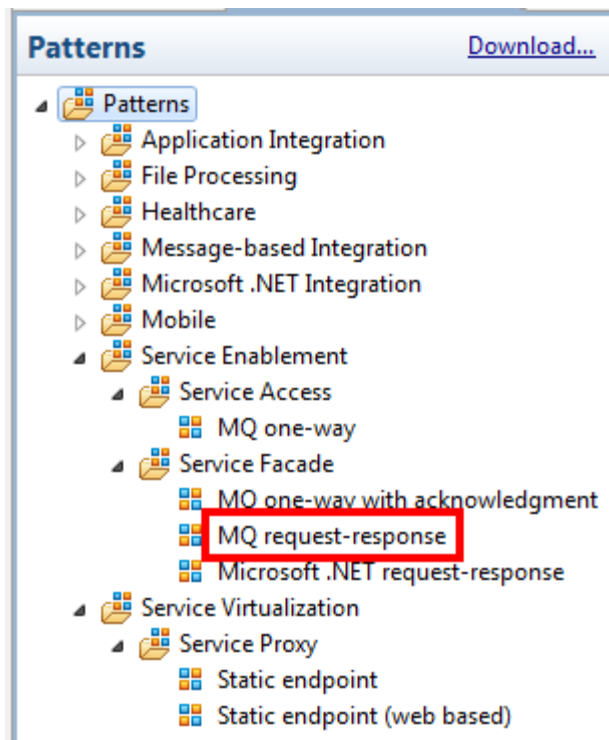
This is the main response flow.



- __1. If necessary, close any open editor views. You should see the Quick Starts wizards list.
- __2. Click **Start from patterns** in the list of **Quick Starts**.



- __3. The **Patterns Explorer** opens in the Navigator.
- __4. Select the **MQ request-response** pattern under **Service Enablement**→**Service Façade**.



5. Take some time to read about the pattern and the solution it will generate.

Service Facade to WebSphere MQ: request-response pattern

Use the Service Facade to WebSphere MQ: request-response pattern to provide a web service facade to functions that are accessible only through WebSphere MQ. This pattern creates a bridge between the synchronous HTTP protocol, which is typically used with web services, and existing applications with WebSphere MQ interfaces that cannot easily be upgraded.

Use this pattern where provider applications provide an XML interface and client applications support calls to web services. The pattern can be extended with transforms to support a service facade to applications with non-XML interfaces over WebSphere MQ.

Solution

The solution is to implement a message flow that provides a service entry point. When a service request is received, the reply identifier is stored on an internal queue and the request is forwarded to the provider application. When the response is received from the provider application, the reply identifier is recovered and a web services reply, which contains the provider response, is returned to the requesting application.

Create New Instance

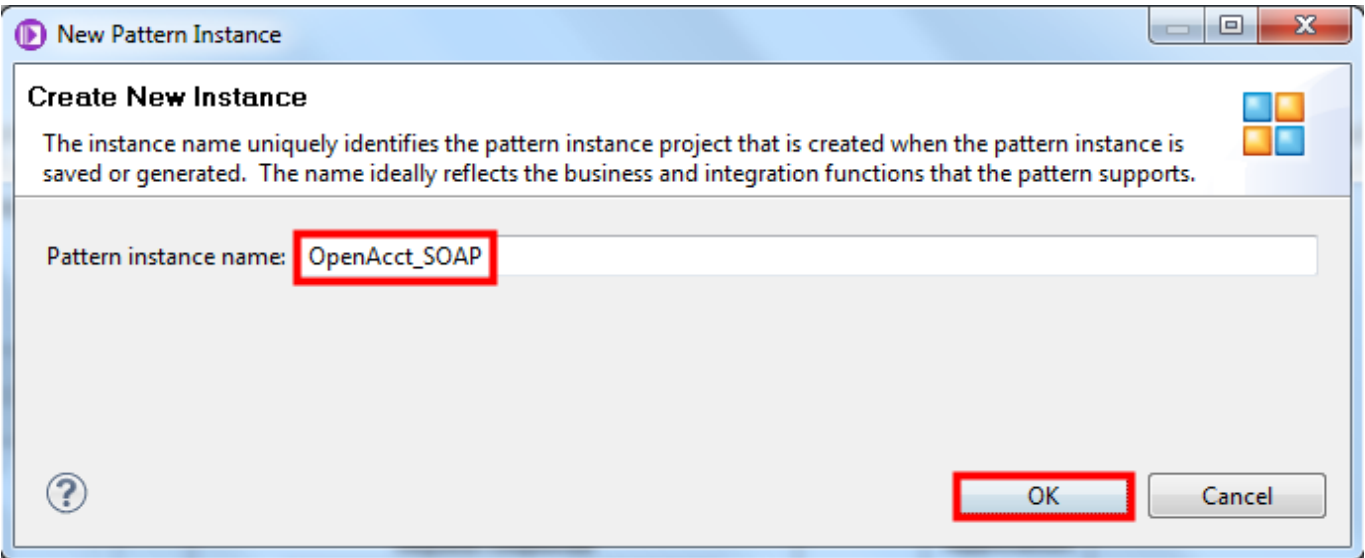
6. Click the **Create New Instance** button to start the Pattern Generation wizard.

Service Facade to WebSphere MQ: request-response pattern

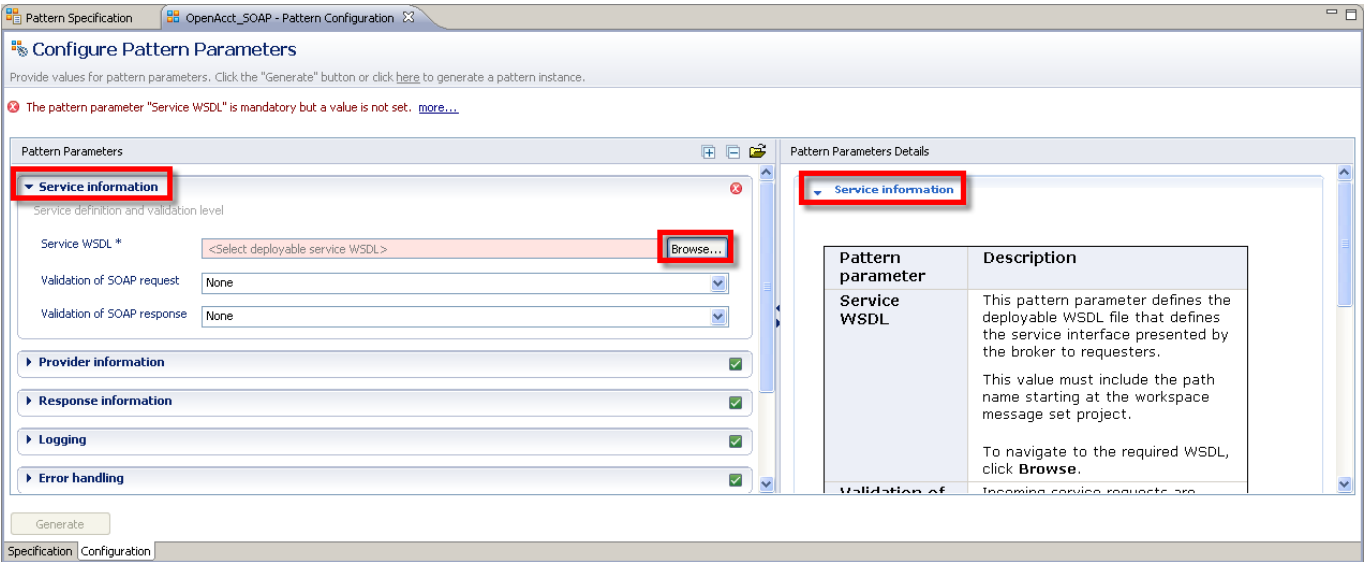
View information about the selected pattern and then click the "Create New Instance" button or click [here](#) to start using a pattern.

Create New Instance

- __7. Enter **OpenAcct_SOAP** as the **Pattern instance name**.
- __8. Click **OK**.

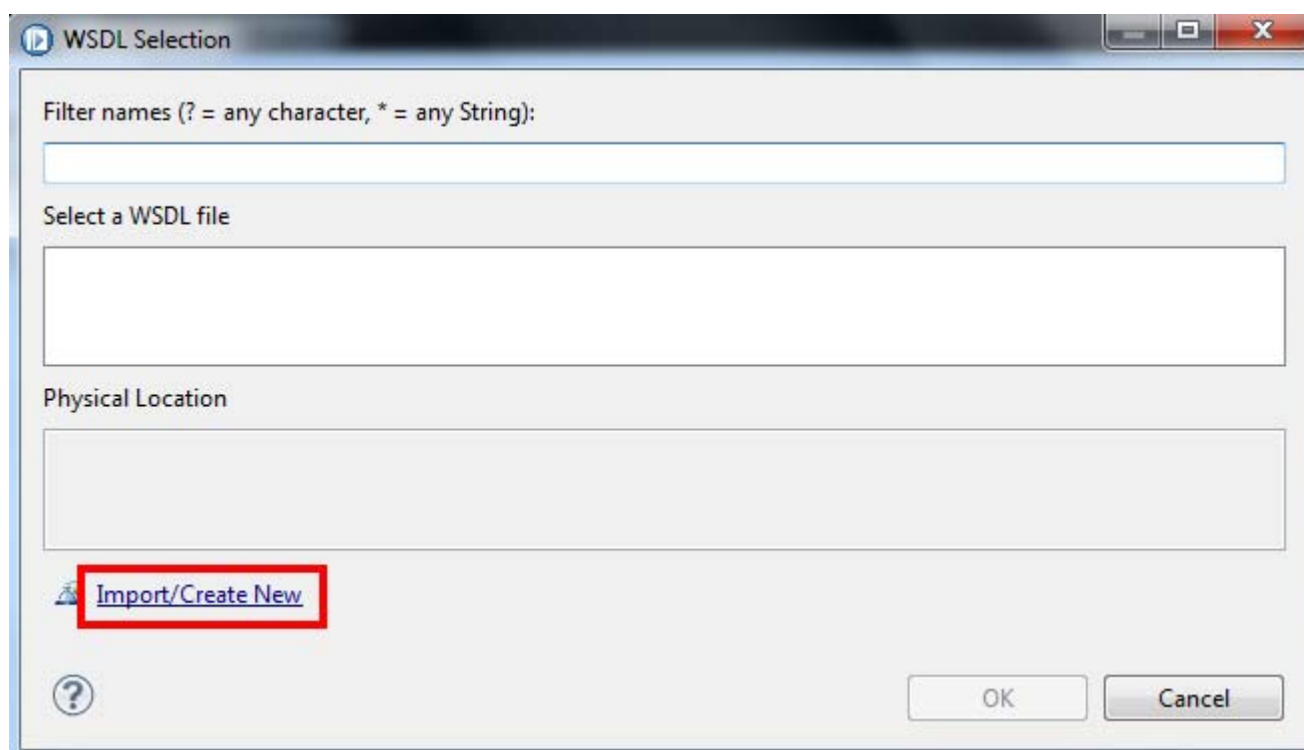


- __9. Expand the **Service Information** section under **Pattern Parameters**. This section is marked with an exception, indicating a required parameter. Also expand the **Service Information** section under **Pattern Parameters Details** to see help for each parameter.
- __10. Click **Browse** to the right of the **Service WSDL** input.



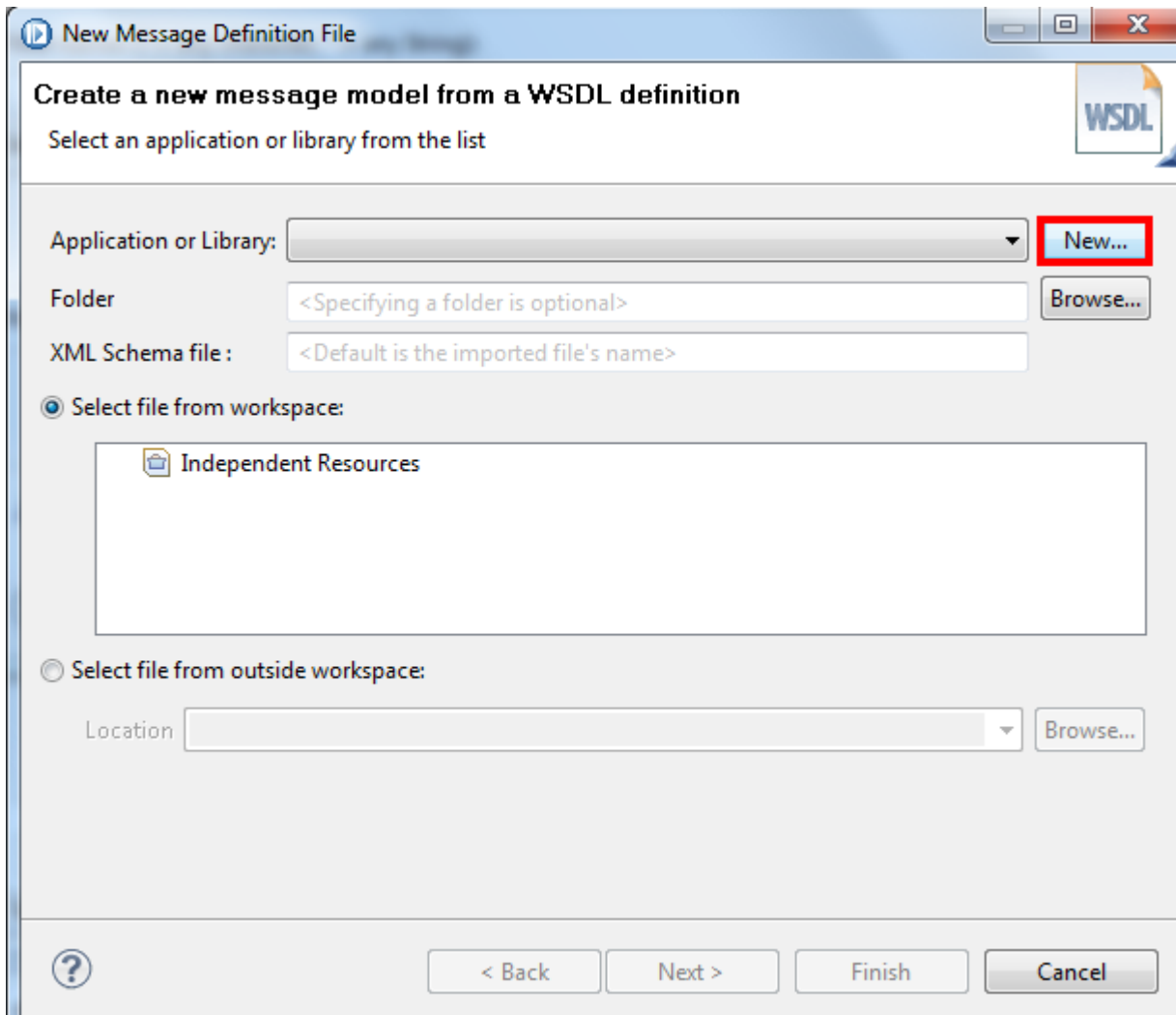
A selection dialog is shown.

__11. Click the **Import/Create New** link.



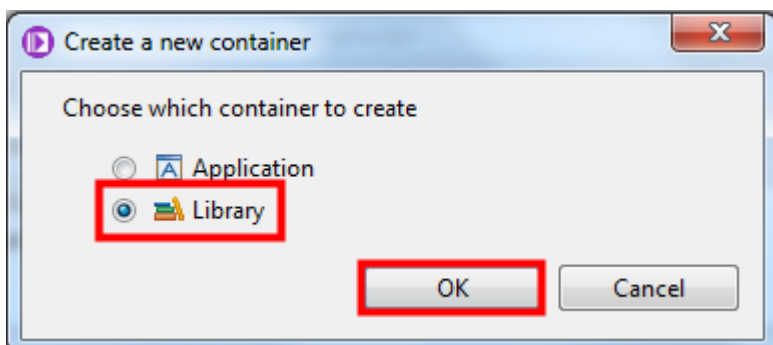
A library will be created to import the WSDL into.

__12. Click **New...** to the right of **Application or Library**.



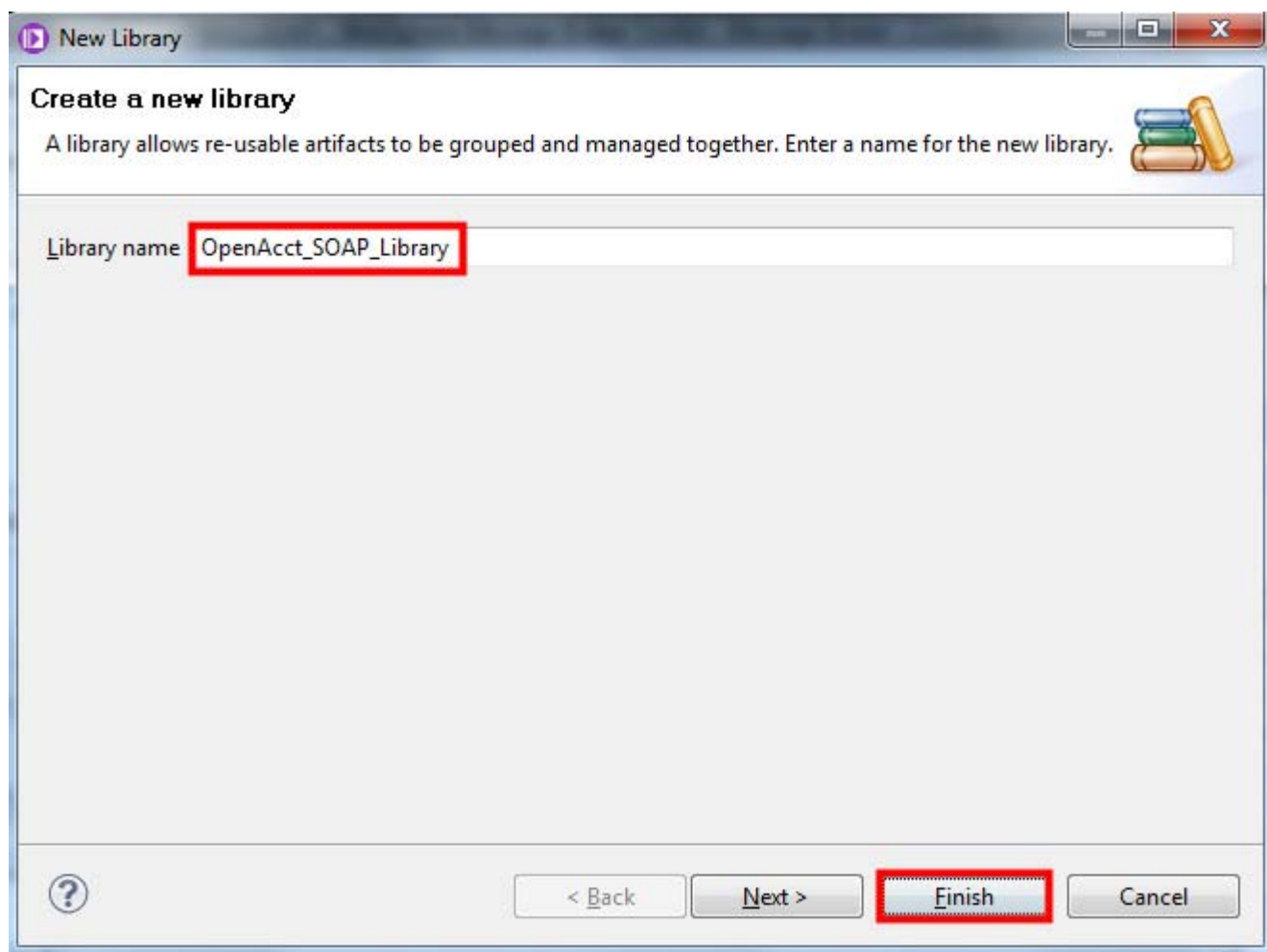
__13. Select the radio button for **Library**.

__14. Click **OK**.



__15. Enter **OpenAcct_SOAP_Library** for **Library name**.

__16. Click **Finish**.



New Library

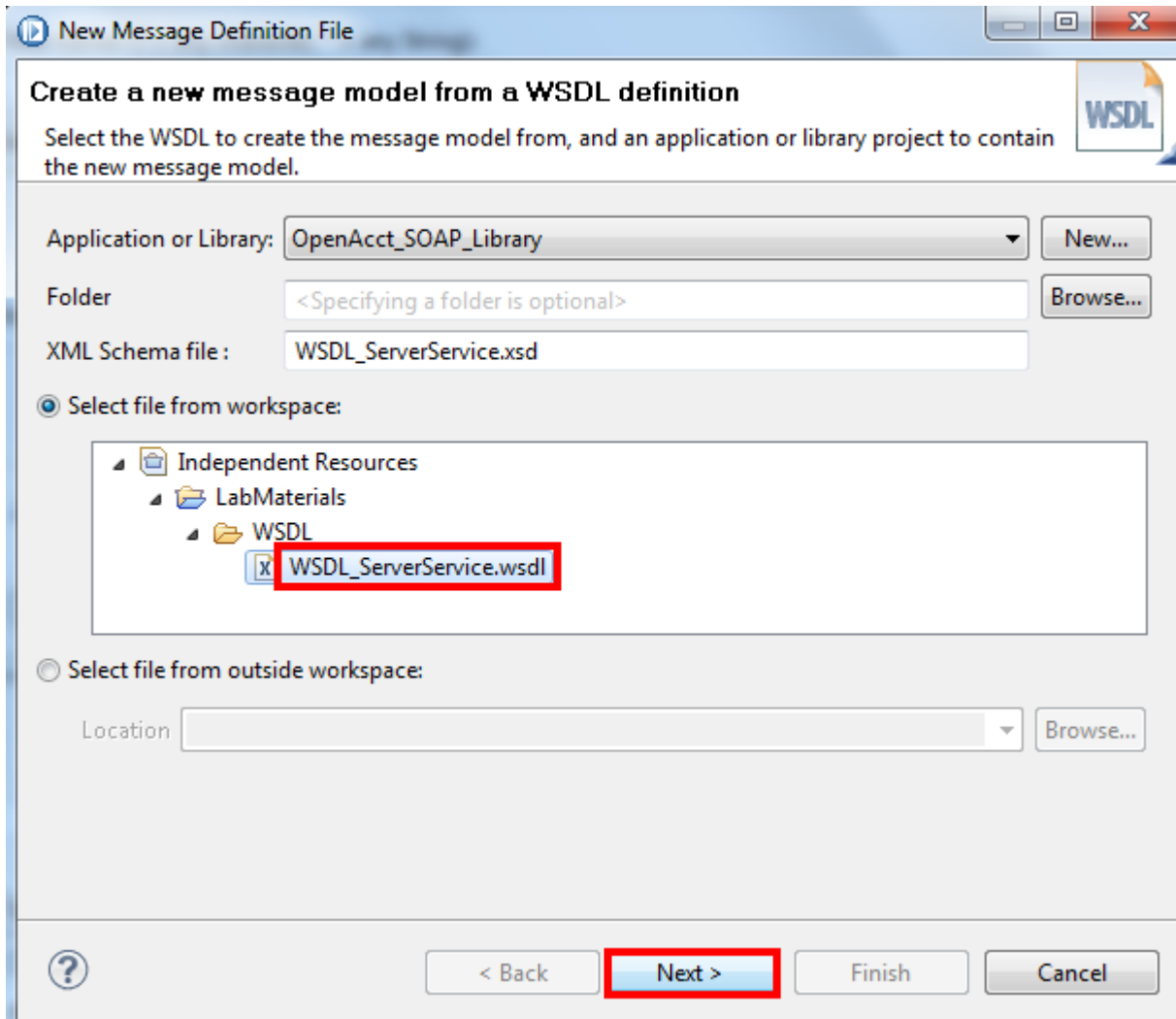
Create a new library

A library allows re-usable artifacts to be grouped and managed together. Enter a name for the new library.

Library name **OpenAcct_SOAP_Library**

? < Back Next > **Finish** Cancel

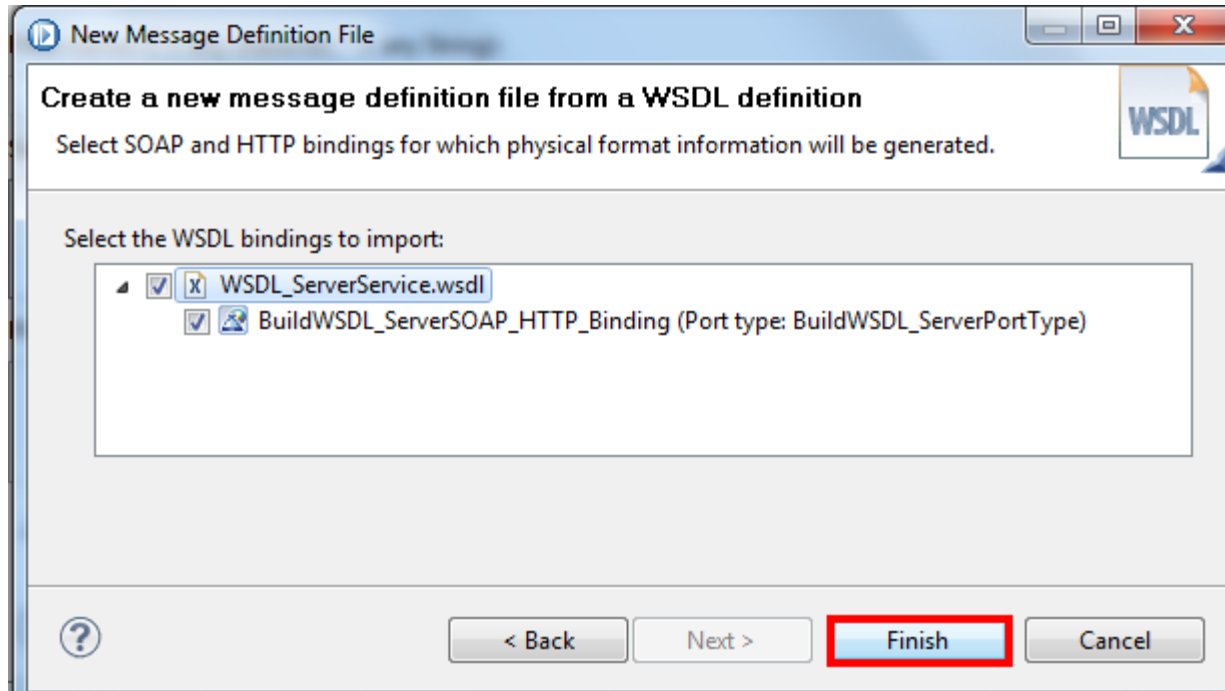
- __17. Expand **Independent Resources**→**LabMaterials**→**WSDL**.
- __18. Select the **WSDL_ServerService.wsdl** resource.
- __19. Press the **Next** button.



A WSDL may have multiple bindings. In this case there is only one.

__20. Click **Finish**.

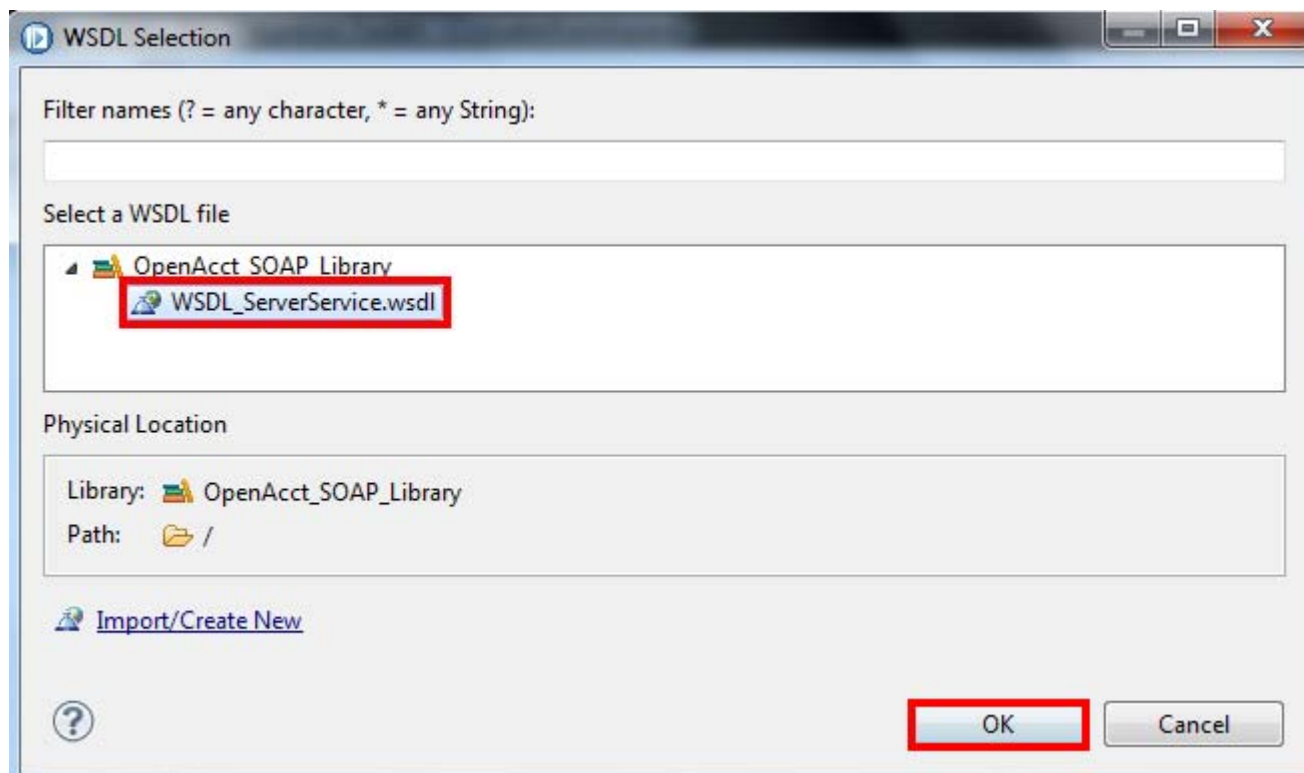
__21. Wait for the wizard to complete.



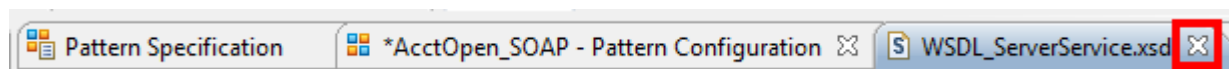
When the wizard finishes several things have been done. A Library has been created (**OpenAcct_SOAP_Library**). The library contains schema definitions for all of the possible inputs to and outputs from the web service that are defined in the WSDL file, including the inclusion of the built-in SOAP schemas.

__22. Highlight the **WSDL_ServerService.wsdl** in the new **OpenAcct_SOAP_Library**.

__23. Click **OK**.



__24. The Import Wizard also opened the **WSDL_ServerService.xsd** schema. Click the **X** to close the schema view.



- __25. The **OpenAcct_SOAP – Pattern Configuration** window should be visible. The **Service WSDL** parameter is now complete.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

Service information ☒

Service definition and validation level

Service WSDL *

Validation of SOAP request

Validation of SOAP response

Provider information ☒

Response information ☒

Logging ☒

Error handling ☒

General ☒

Specification Configuration

Pattern Parameters Details

Service information

Response information

Provider information

Logging

Error handling

General

- __26. Collapse the **Service information** section.
- __27. Expand the **Provider Information** section.
- __28. Change the **Provider request queue** field to **IN** (be sure to use uppercase – recall that queue names are case-sensitive).

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

Service information ☒

Provider information ☒

Provider address

Provider queue manager

Provider request queue *

Response information ☒

Logging ☒

Error handling ☒

General ☒

Pattern Parameters Details

Service information

Response information

Provider information

Pattern parameter	Description
Provider queue manager	This pattern parameter identifies the queue manager on which the provider application receives requests from the broker message flow. If left blank, the default is the broker queue manager.
Provider request queue	This pattern parameter identifies the queue on which the provider application receives requests from the broker message flow.

- __29. Expand the **Response Information** section.
- __30. Change the **Response queue** field to **REPLY**.
- __31. Change the **Store queue** field to **STATE**.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

- Service information** ✓
- Provider information** ✓
 - Provider address
 - Provider queue manager
 - Provider request queue * IN
- Response information** ✓
 - Response queue and validation requirements
 - Response queue * **REPLY**
 - Store queue * **STATE**
 - Expiry of store queue messages in tenths of a second * No timeout
- Logging** ✓
- Error handling** ✓
- General** ✓

Pattern Parameters Details

- Service information**
- Response information**

Pattern parameter	Description
Response queue	This pattern parameter identifies the broker queue to which the provider application must return responses.
Store queue	This pattern parameter defines the queue that is used to store the MQMD headers while waiting for a provider response.
Expiry of store queue messages in tenths of a second	<p>This pattern parameter defines the expiry time, in tenths of a second, for messages that hold the HTTP reply identifier.</p> <p>This value must exceed the maximum expected response time.</p> <p>You can enter your own value, or accept the default value <code>No timeout</code>.</p>
- Provider information**

- __32. Collapse the **Provider information** and **Response information** sections.
- __33. Expand the **Logging** section.
- __34. Select the **Logging required** check box.
- __35. Accept the default value for the **Log queue (LOG)**.

Pattern Specification *OpenAcct_SOAP - Pattern Configuration

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

- Service information
- Provider information
- Response information
- Logging**
- Error handling
- General

Pattern Parameters Details

- Service information
- Response information
- Provider information
- Logging**

Pattern parameter	Description
Logging required	<p>This pattern parameter determines whether the pattern instance includes the code for logging.</p> <p>If <i>Logging required</i> is cleared, the Log subflow is not included.</p> <p>If <i>Logging required</i> is selected, a Log subflow is included in the m</p>

- __36. Collapse the **Logging** section.
- __37. Expand the **Error Handling** section.

Basic error handling is included by default. There is nothing to change in this section.

Pattern Specification *OpenAcct_SOAP - Pattern Configuration

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

- Service information
- Provider information
- Response information
- Logging
- Error handling**
- General

Pattern Parameters Details

- Logging
- Error handling**

Pattern parameter	Description
Error message required	<p>This pattern parameter defines whether the pattern application creates the message flow elements to create error messages.</p> <p>If <i>Error message required</i> is selected, values must be set for <i>Error queue manager</i> and <i>Error queue</i>.</p>
Error queue manager	<p>This pattern parameter defines the queue manager for error messages. It can be left blank if the broker queue manager is used for logging.</p>
Error queue	<p>This pattern parameter defines the queue for error messages.</p> <p>This parameter is required only if <i>Error message required</i> is selected.</p>

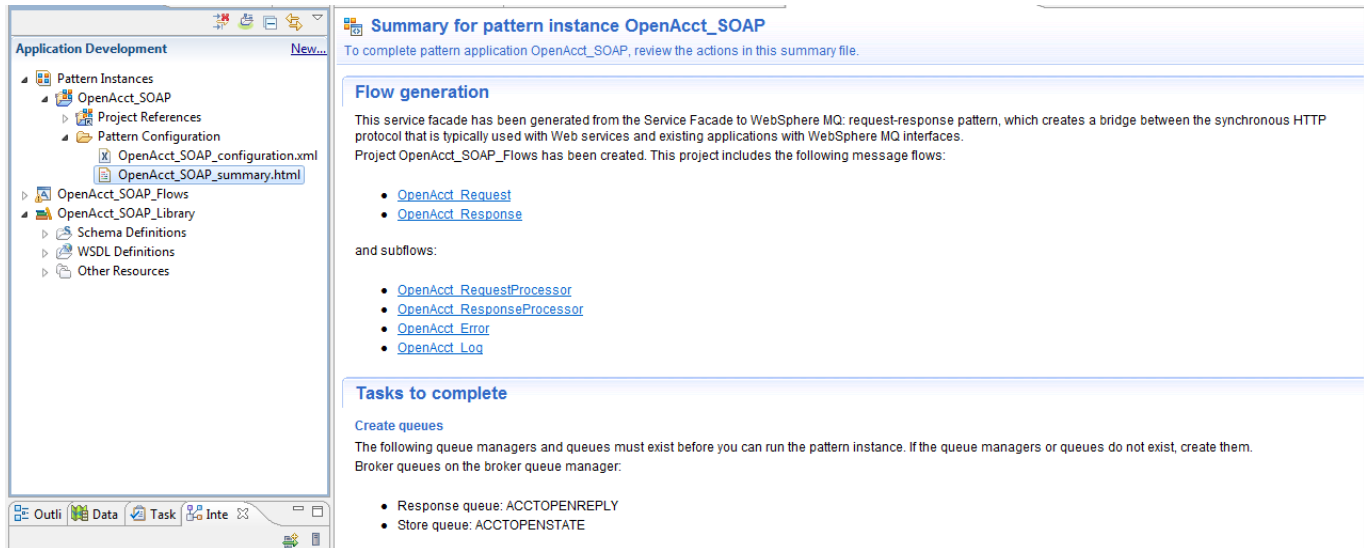
- __38. Collapse the **Error Handling** section.
- __39. Expand the **General** section.
- __40. Enter **OpenAcct_** in the **Flow prefix** field. **Please note the trailing underscore!**
- __41. Enter **ACCTOPEN.** in the **Queue prefix** field. **Please do not forget the period at the end of the prefix.**
- __42. Make sure that the entries are in the **prefix** fields.
- __43. Click '**Generate**' to build the pattern.

The screenshot shows the 'Pattern Parameters' dialog box with the 'General' section expanded. The 'General' section is highlighted with a red box. Below it, the 'Specify naming conventions and description' section contains several input fields. The 'Flow prefix' field is highlighted with a red box and contains the text 'OpenAcct_'. The 'Queue prefix' field is also highlighted with a red box and contains the text 'ACCTOPEN.'. The 'Generate' button at the bottom left is also highlighted with a red box. The 'Specification' tab is selected at the bottom of the dialog.

Pattern Parameters	
Response information	✓
Logging	✓
Error handling	✓
General	✓
Specify naming conventions and description	
Broker schema	mqsi
Flow prefix	OpenAcct_
Flow suffix	
Queue prefix	ACCTOPEN.
Queue suffix	
Short description	
Long description	

Generate

Specification | Configuration



When the wizard has completed, you will see the **Summary for pattern instance OpenAcct_SOAP** window, which provides a brief summary of the artifacts that have been created and some further tasks that must be completed, such as creation of the necessary queues.

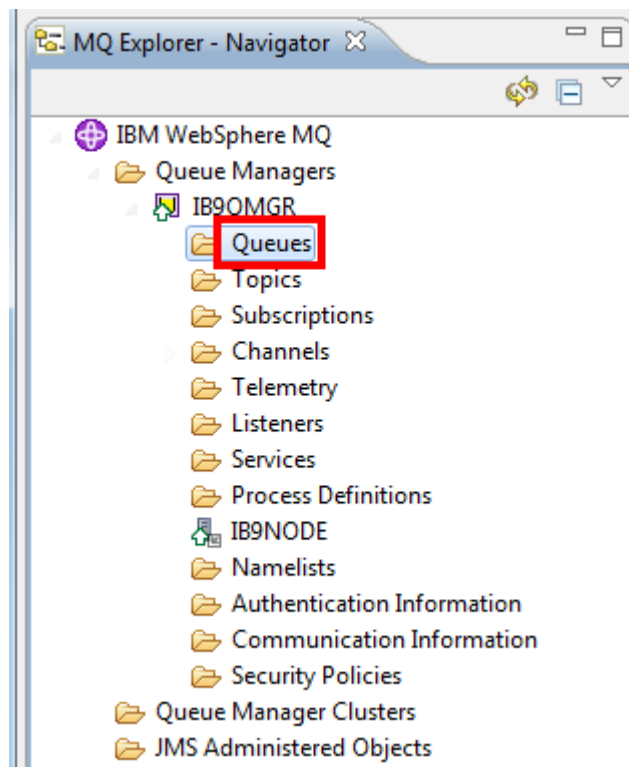
The wizard has created an Application project called **OpenAcct_SOAP_Flows**. This application contains two message flows, **OpenAcct_Request** and **OpenAcct_Response**, plus four sub flows used by these message flows, **OpenAcct_RequestProcessor**, **OpenAcct_ResponseProcessor**, **OpenAcct_Error**, and **OpenAcct_Log**.

You also will see the **OpenAcct_SOAP** Pattern Instance in your Application Development view. Under the **Pattern Configuration** folder, you will see the `OpenAcct_SOAP_configuration.xml` file, the saved pattern configuration, and the `OpenAcct_SOAP_summary.html` file, the summary screen, that can be used for later reference or pattern regeneration.

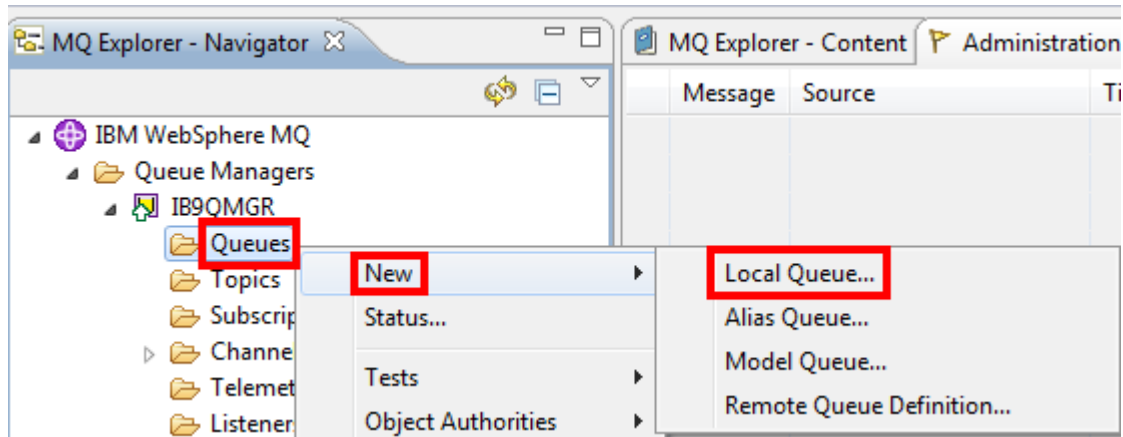
The queues will now be created using MQ Explorer. The ACCTOPEN.IN queue already exists. It is used by the existing MQ-based account open service. In a real-life scenario it is likely that the existing service would be running on a different system, in which case a remote queue definition would have to be made on the system that the node runtime is running on. It is possible to modify the MQOutput node that is used to send the message to route the message directly to the remote system by specifying a queue manager name. It is a better practice to keep the location of remote applications external from the applications themselves, including message flows.

__44. Switch to MQ Explorer.

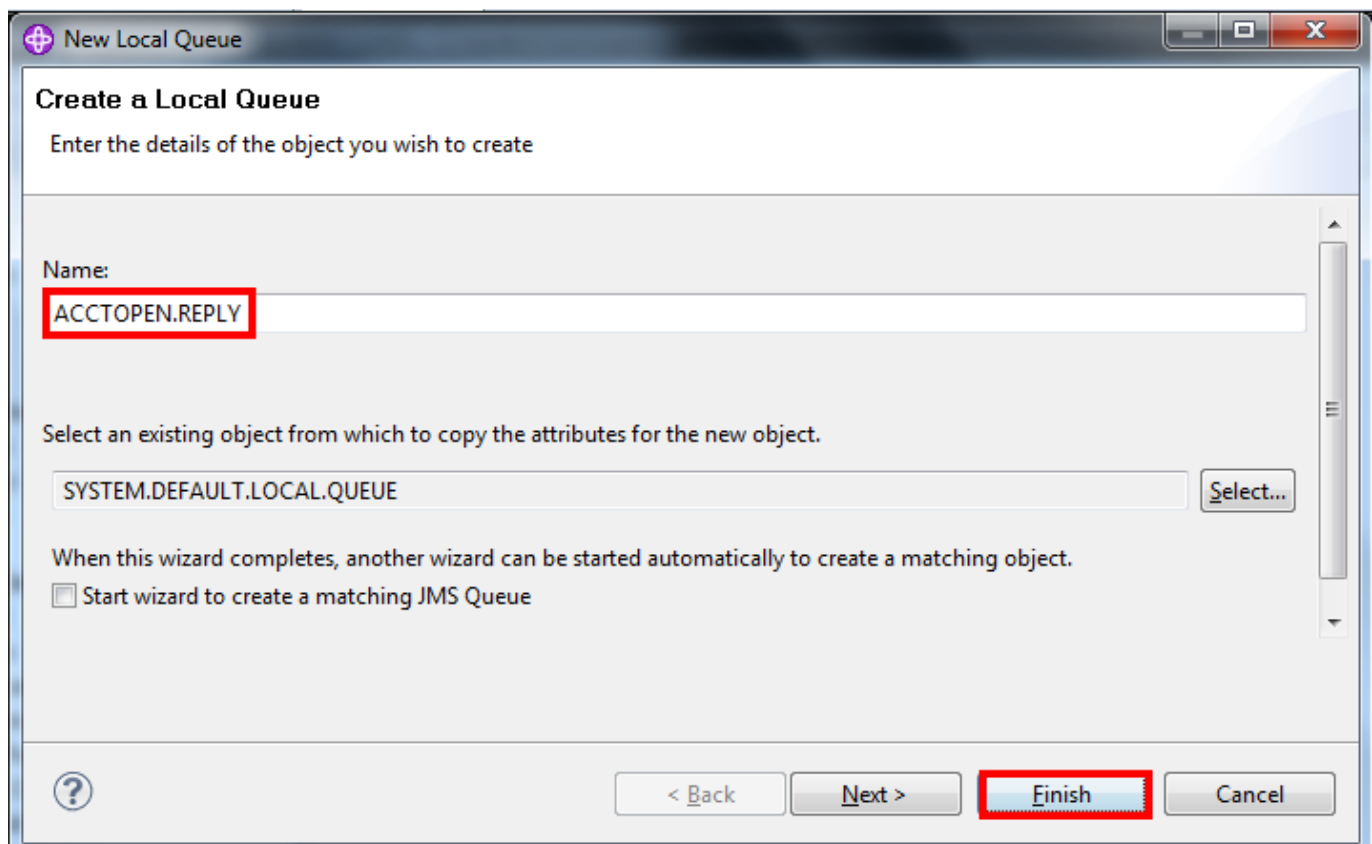
__45. If necessary, expand **Queue Managers**→**IB9QMGR** so that **Queues** is visible.



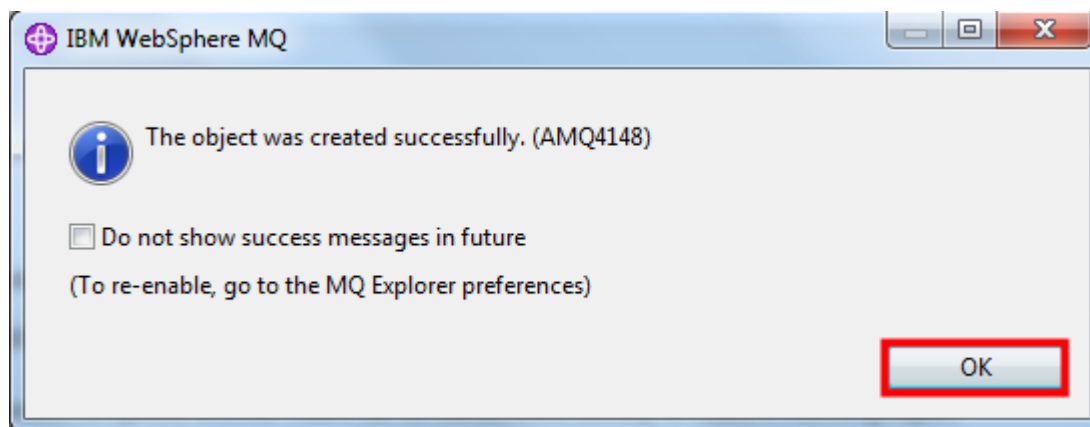
- __46. Select **Queues**.
- __47. Press the right mouse button.
- __48. Select **New→Local Queue** from the menu.



- __49. Enter **ACCTOPEN.REPLY** as the **name** of the new queue.
- __50. Press the **Finish** button to create the queue.



__51. Press the **OK** button to dismiss the notification.



__52. Repeat the previous three steps to create the following queues:

- **ACCTOPEN.STATE**
- **ACCTOPEN.LOG**
- **ACCTOPEN.ERROR**

__53. Minimize the MQ Explorer window.

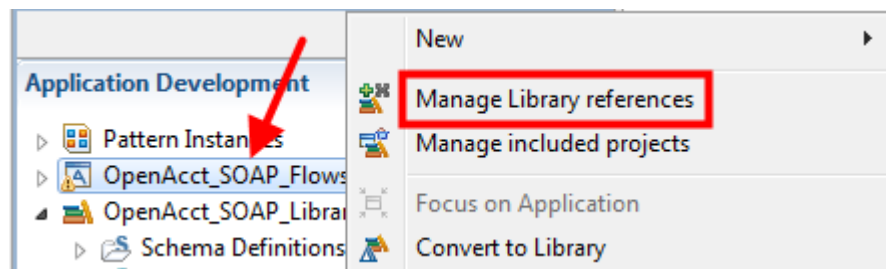
__54. Return to the Integration Toolkit and close the **OpenAcct_SOAP_summary** and **OpenAcct_SOAP – Pattern Configuration** tabs.



__55. Select the new **OpenAcct_SOAP_Flows** application.

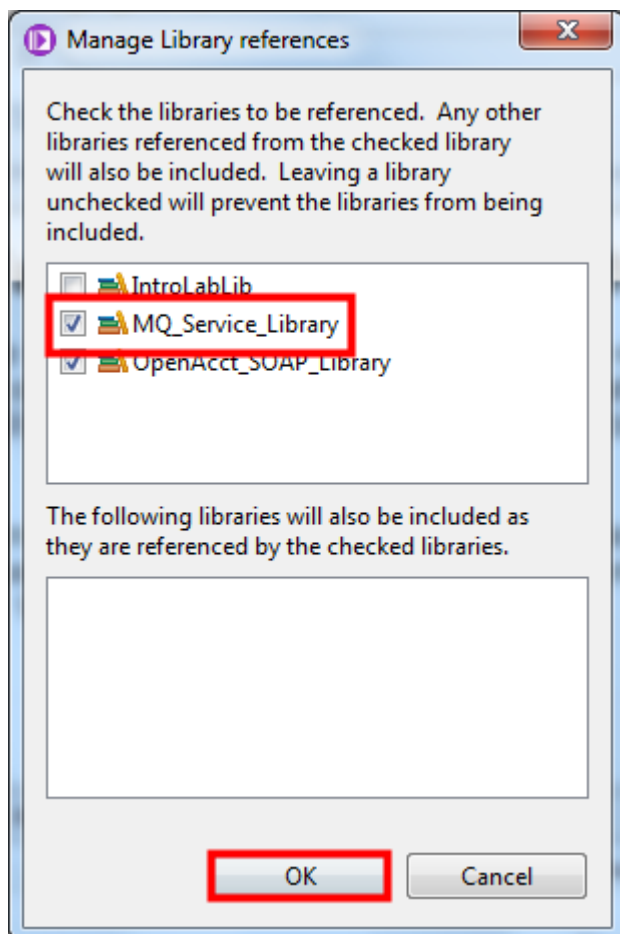
__56. Press the right mouse button.

__57. Select **Manage Library references** from the menu.

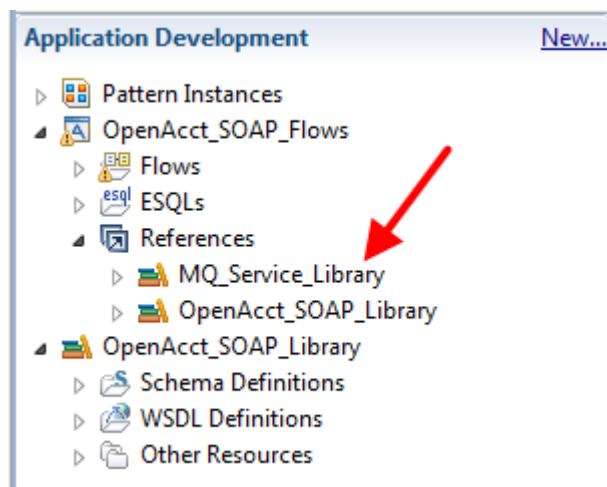


__58. Select the **MQ_Service_Library**.

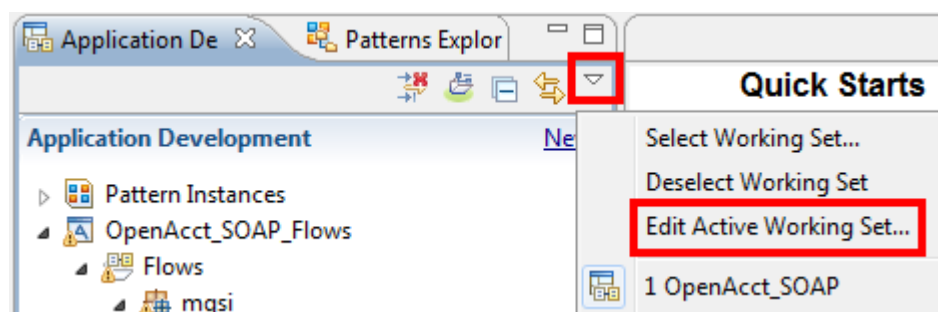
__59. Click **OK**.



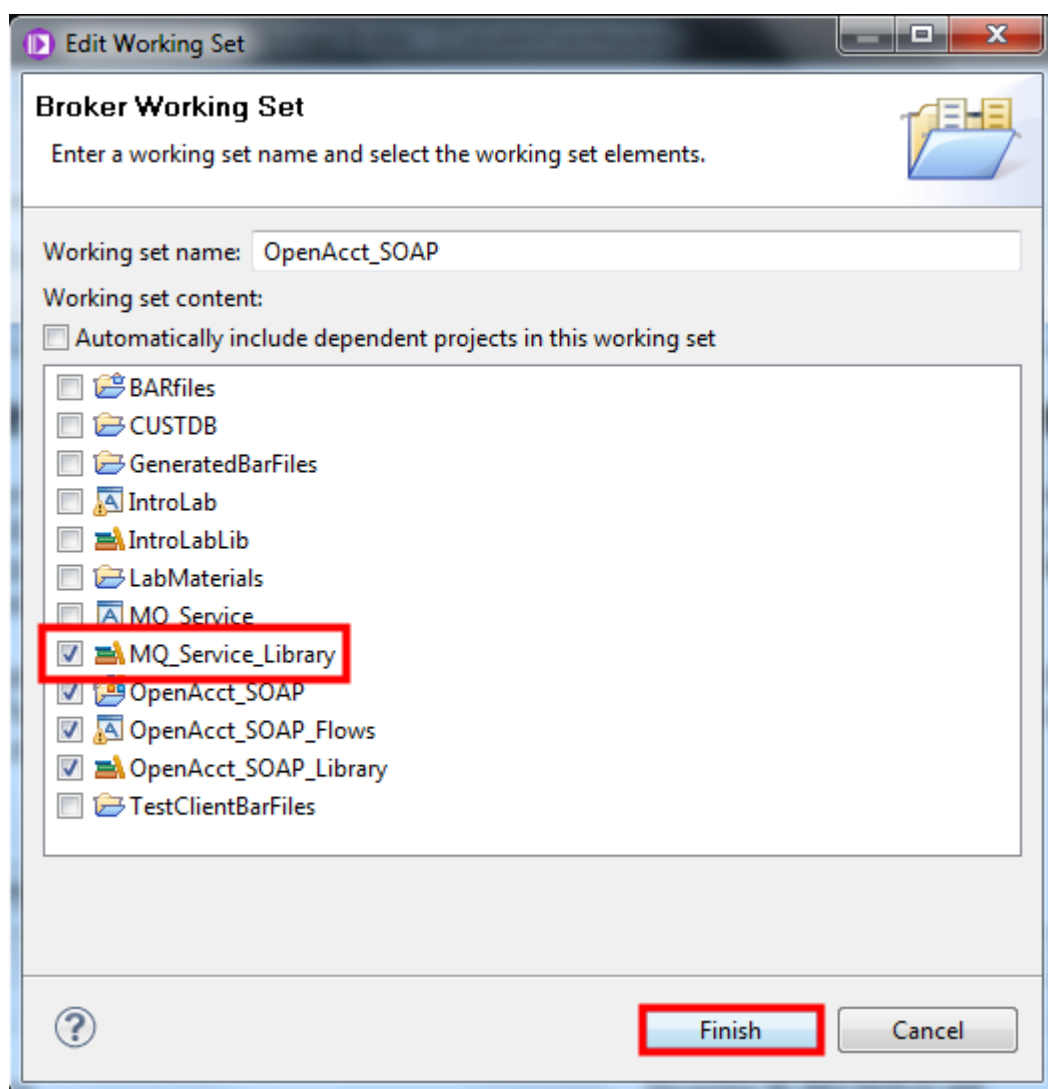
__60. You will see that the **MQ_Service_Library** library has been added to the **References**.



- __61. Click the small triangular icon on the project navigator in the **Application Development** tab.
- __62. Select '**Edit Active Working Set ...**' from the menu.



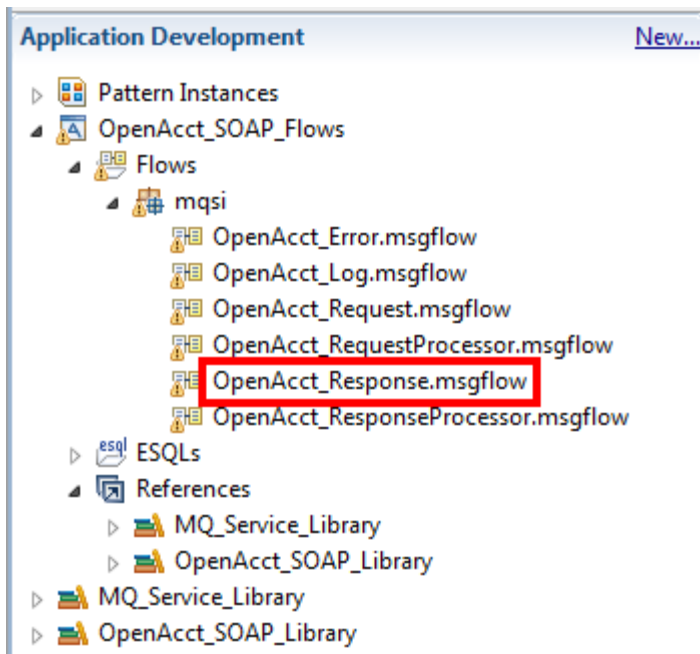
- __63. Select the **MQ_Service_Library** check box. Do not remove any selections.
- __64. Click **Finish**.



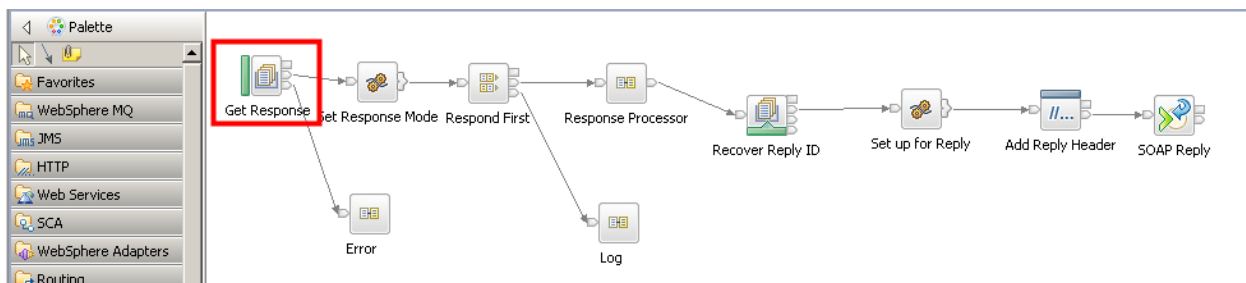
The response message flow needs to be configured for the message format sent by the MQ back-end service.

__65. Expand **OpenAcct_SOAP_Flows**→**Flows**→**mqsi**.

__66. Double Click the **OpenAcct_Response.msgflow**.



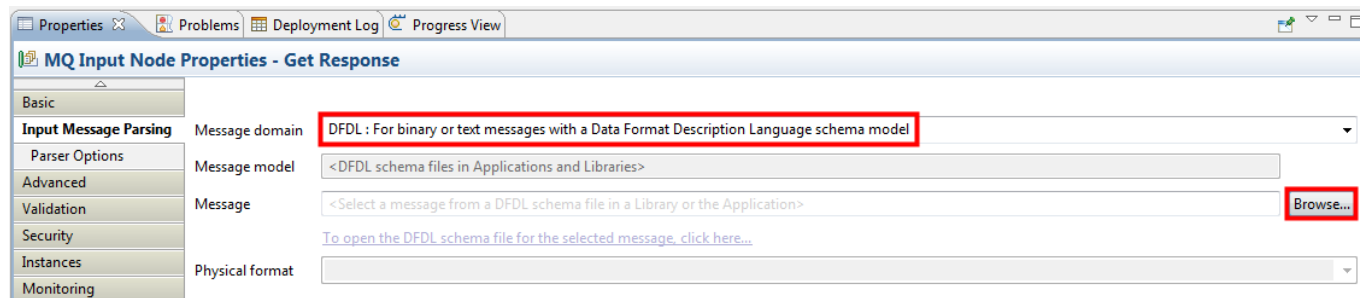
__67. Select the **Get Response** input node.



__68. Select the **Input Message Parsing** tab in the **Properties** view.

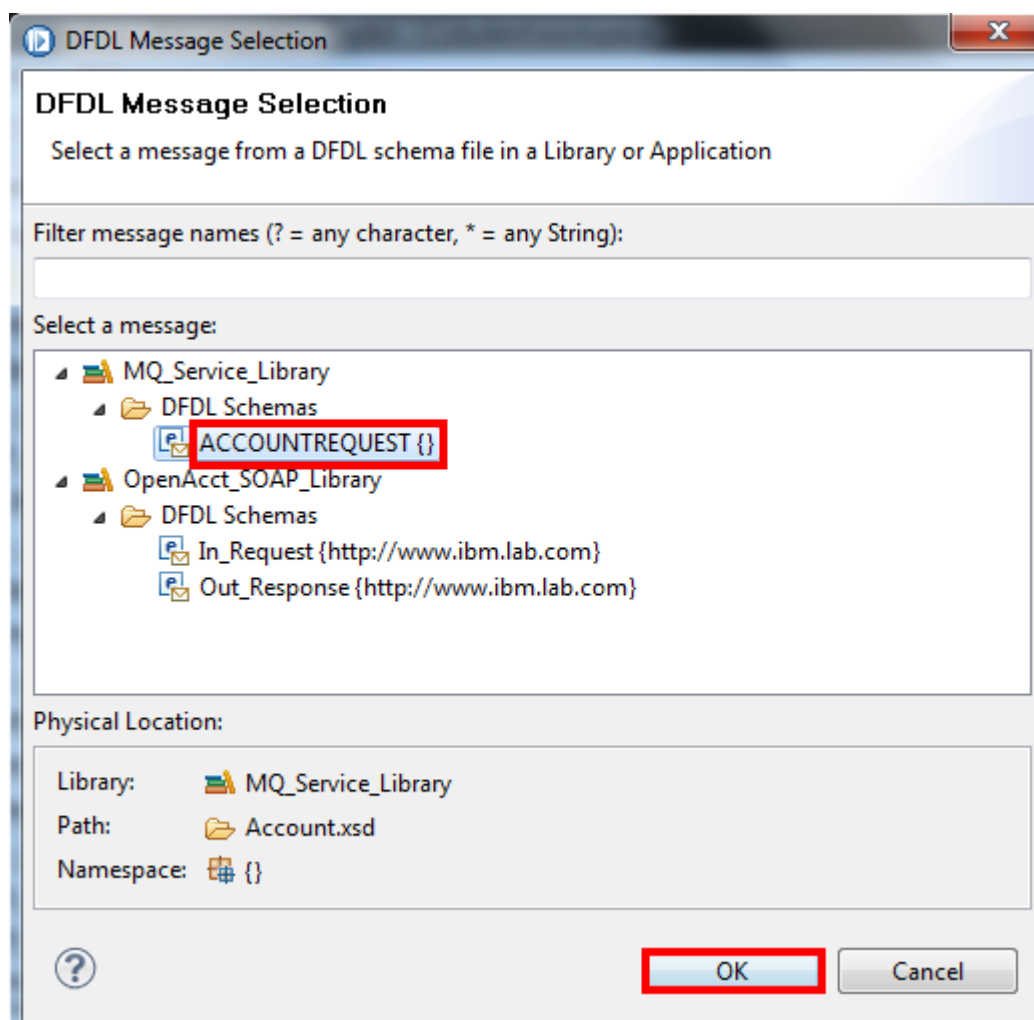
__69. Select **DFDL** for **Message domain** using the drop-down menu.

__70. Press the **Browse** button next to **Message**.

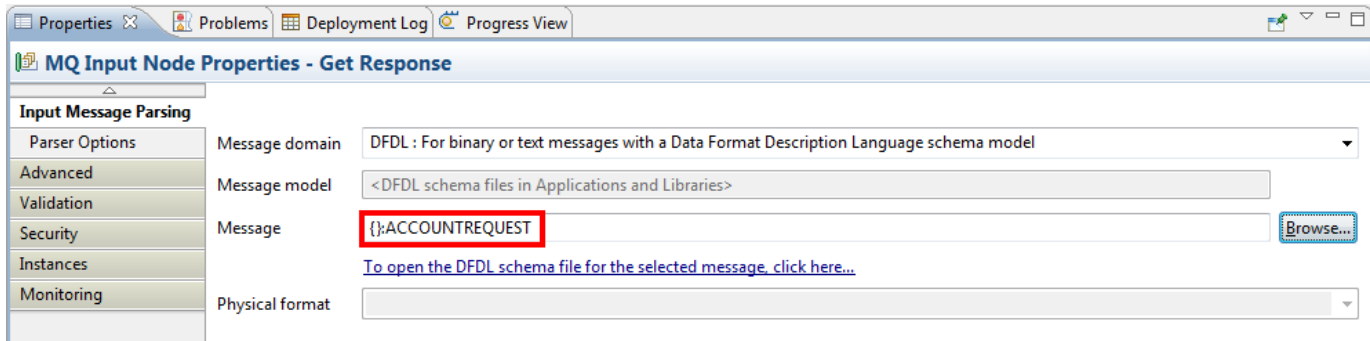


__71. Select the **ACCOUNTREQUEST{}** message definition under **MQ_Service_Library**.

__72. Press the **OK** button.



__73. The message should now be visible in the node properties.



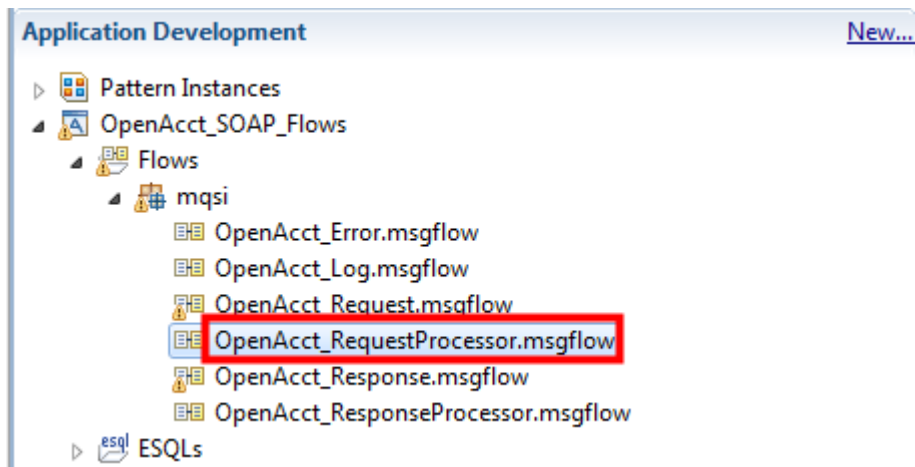
__74.  Save the flow (**Ctrl+S**).

__75. Close the Message Flow editor.

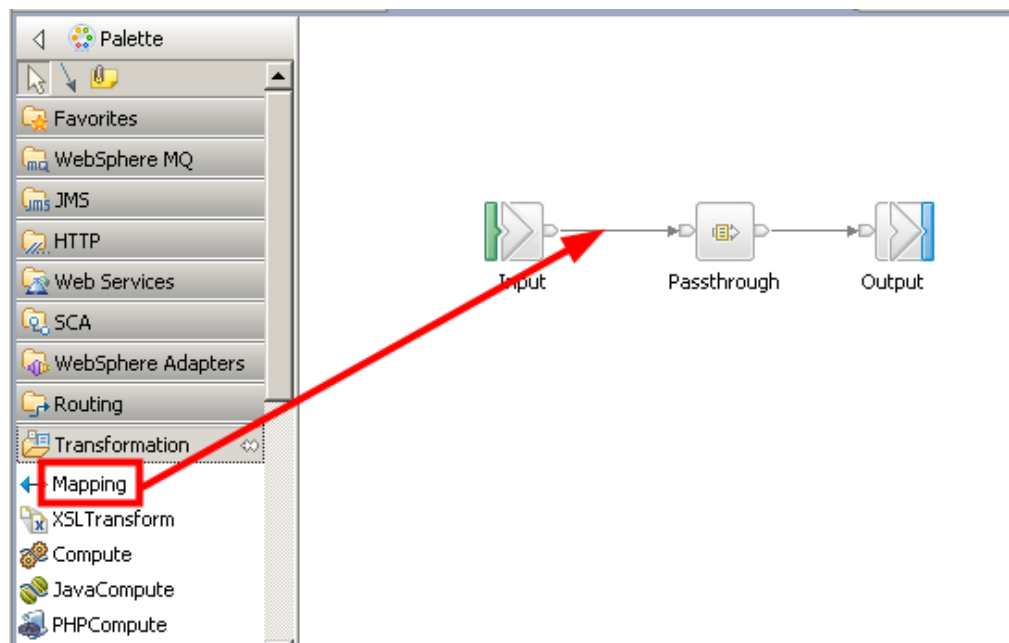


Now you need to create the mappings between the web services message format and the MQ back-end service format. These mappings will be placed in the **OpenAcct_RequestProcessor** and **OpenAcct_ResponseProcessor** sub flows.

__76. Double click the **OpenAcct_RequestProcessor.msgflow** message flow.

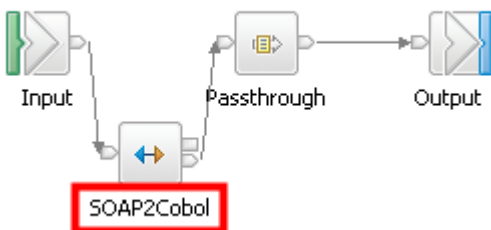


- __77. Expand the **Transformation** drawer.
- __78. Drag a Mapping node from the palette directly on the connector between the **Input node** and the **Passthrough node**.
- __79. Drop the node when the connector color changes to blue.

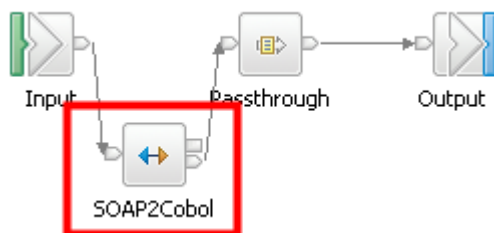


The Mapping node should now have been connected with both existing nodes.

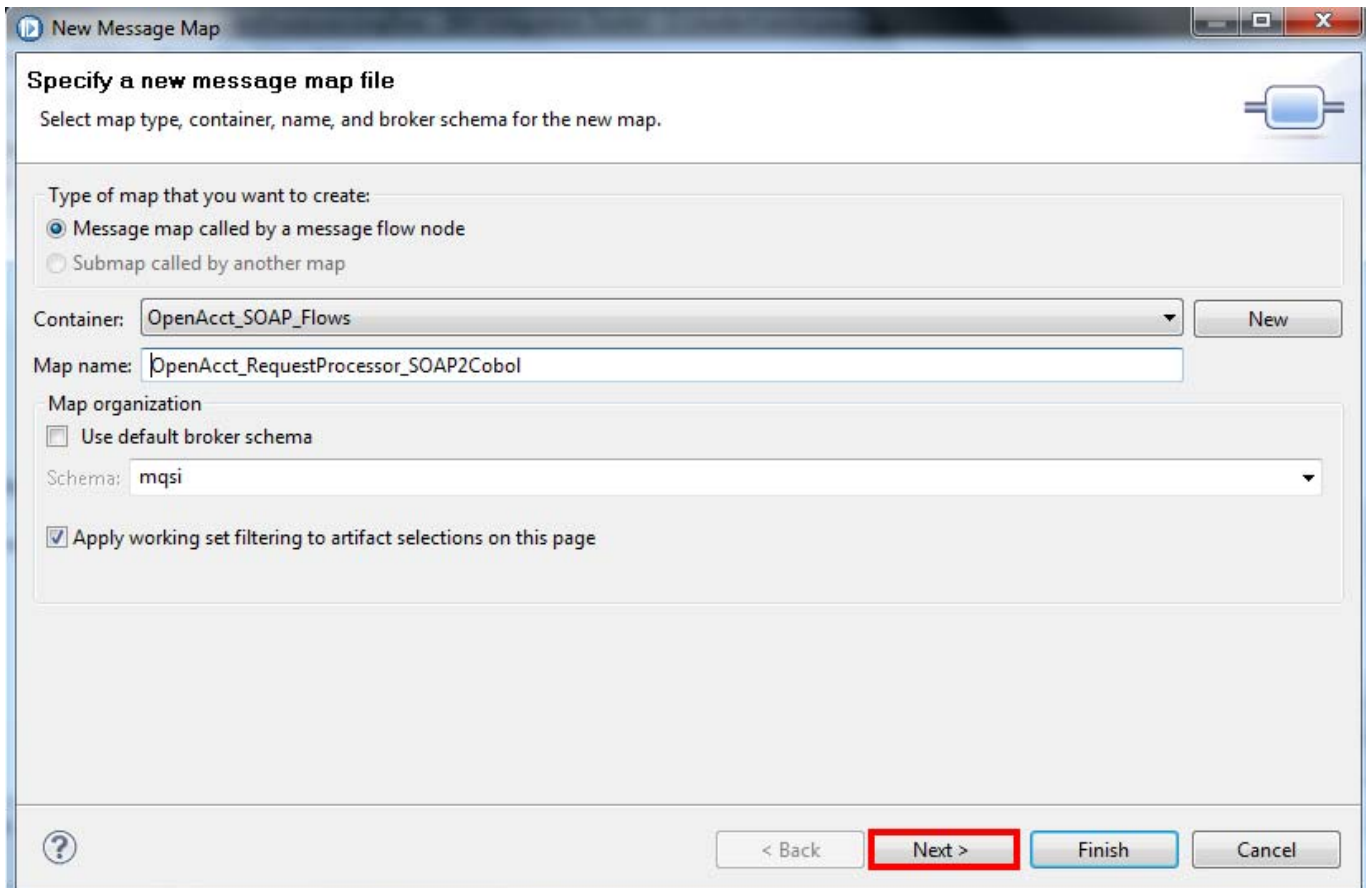
- __80. Change the name of the mapping node to **SOAP2Cobol**.



- __81. Double click the new mapping node (**SOAP2Cobol**).



- __82. Accept the default values for the map name and location.
- __83. Press the **Next** button to continue.

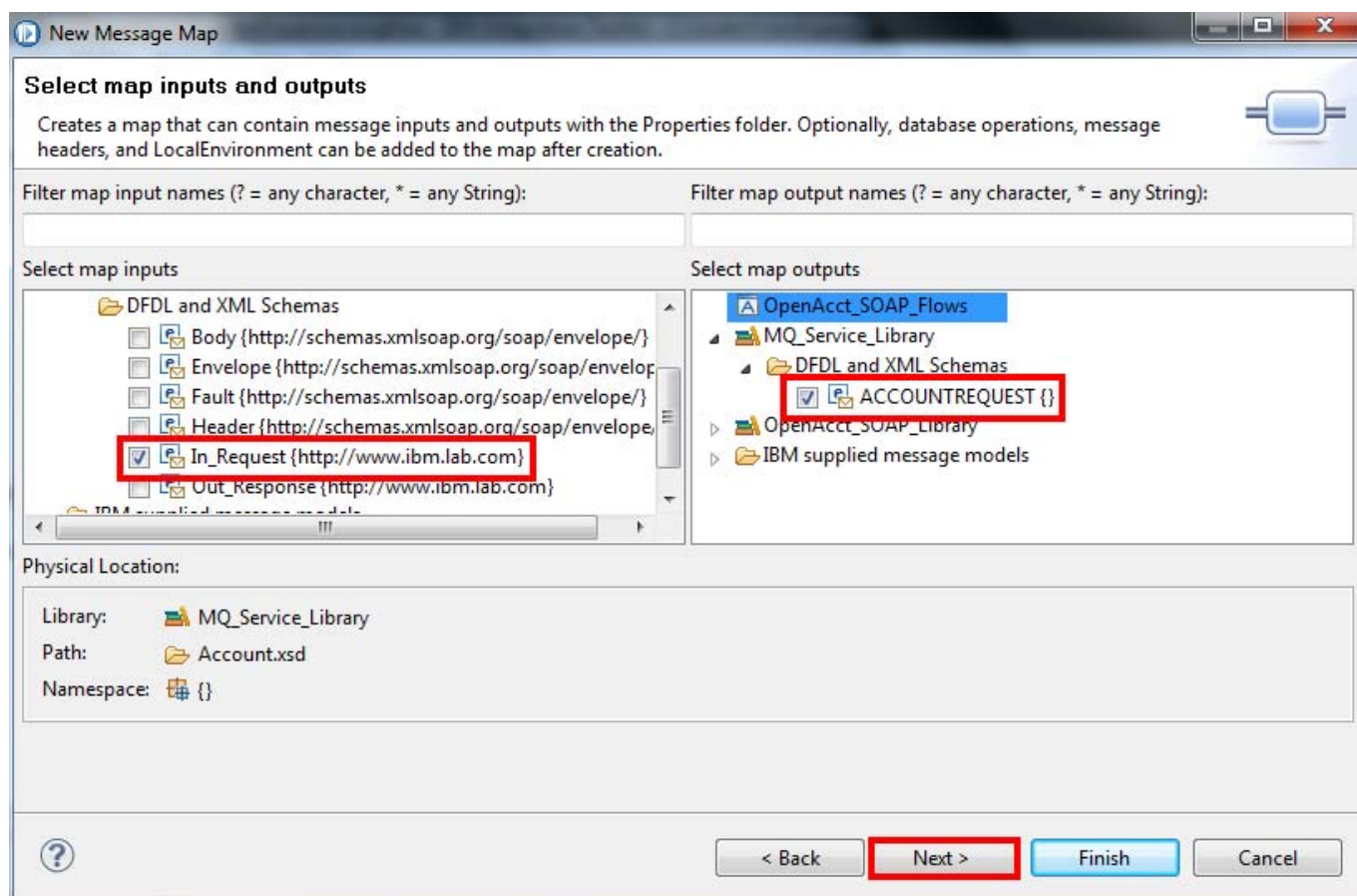


The image shows a 'New Message Map' dialog box with the title 'Specify a new message map file'. Below the title is the instruction 'Select map type, container, name, and broker schema for the new map.' The dialog contains several fields and options:

- Type of map that you want to create:** Two radio buttons are present. The first, 'Message map called by a message flow node', is selected. The second is 'Submap called by another map'.
- Container:** A dropdown menu showing 'OpenAcct_SOAP_Flows' with a 'New' button to its right.
- Map name:** A text field containing 'OpenAcct_RequestProcessor_SOAP2Cobol'.
- Map organization:** A section containing a checkbox 'Use default broker schema' which is unchecked.
- Schema:** A dropdown menu showing 'mqsi'.
- Apply working set filtering to artifact selections on this page:** A checkbox which is checked.

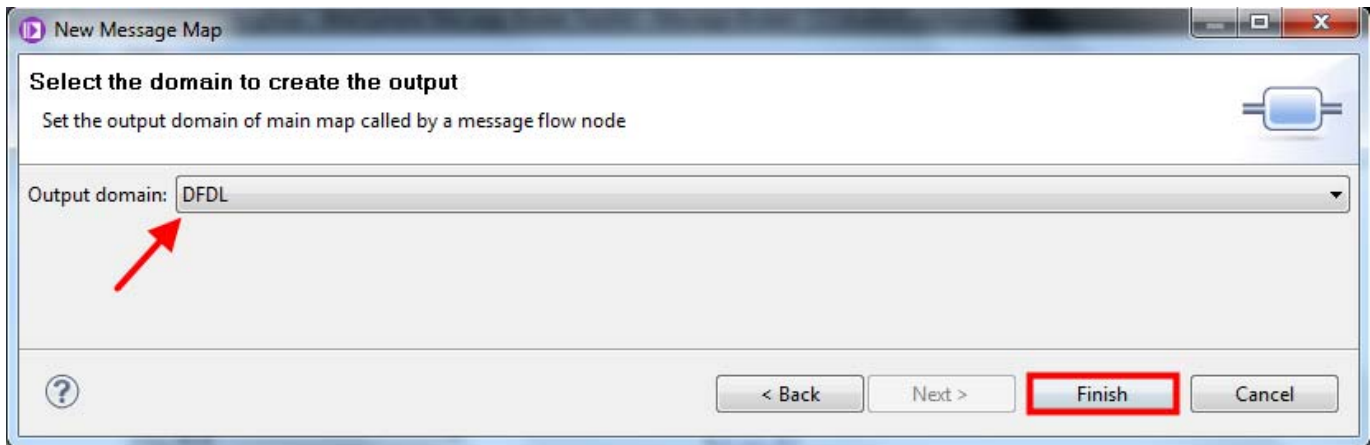
At the bottom of the dialog, there are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a red rectangular border.

- __84. Expand the **OpenAcct_SOAP_Library**→**DFDL and XML Schemas** folder in the **map inputs**.
- __85. Select **In_Request** as map input.
- __86. Expand the **MQ_Service_Library**→**DFDL and XML Schemas** folder in the **map outputs**.
- __87. Select **ACCOUNTREQUEST** as map output.
- __88. Press the **Next** button to continue.



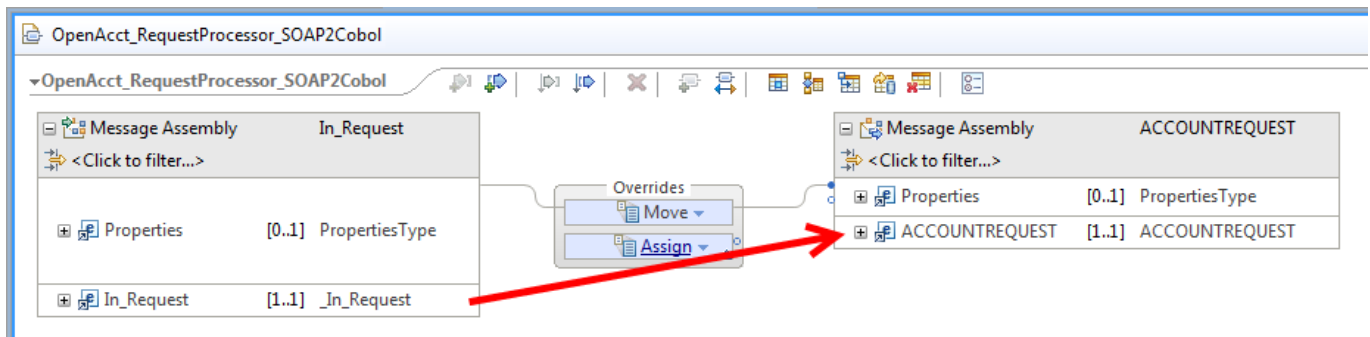
__89. Confirm that the **Output domain** is set to **DFDL**.

__90. Press the **Finish** button to create the map.



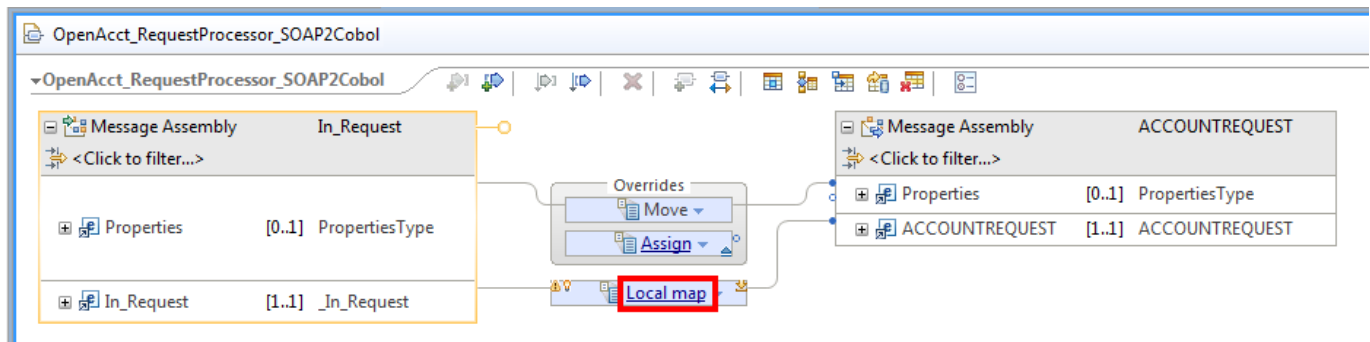
The Mapping editor will open.

__91. Drag the **_In_Request** source to the **ACCOUNTREQUEST** target.



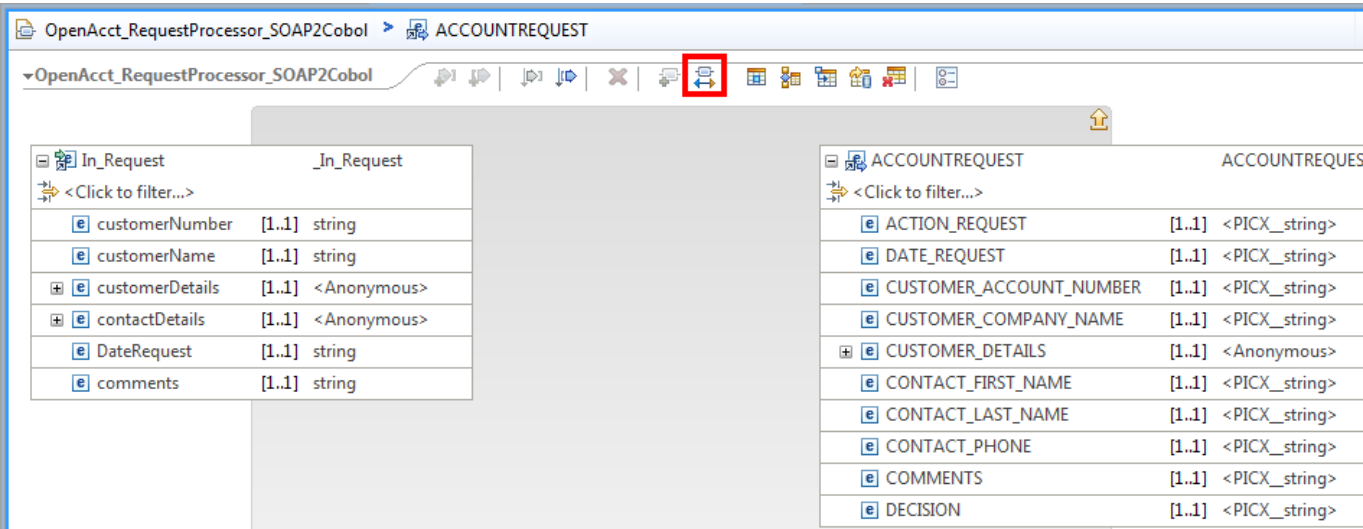
The source and target are complex structures. A local mapping operation will be generated.

__92. Click the **Local map** link.

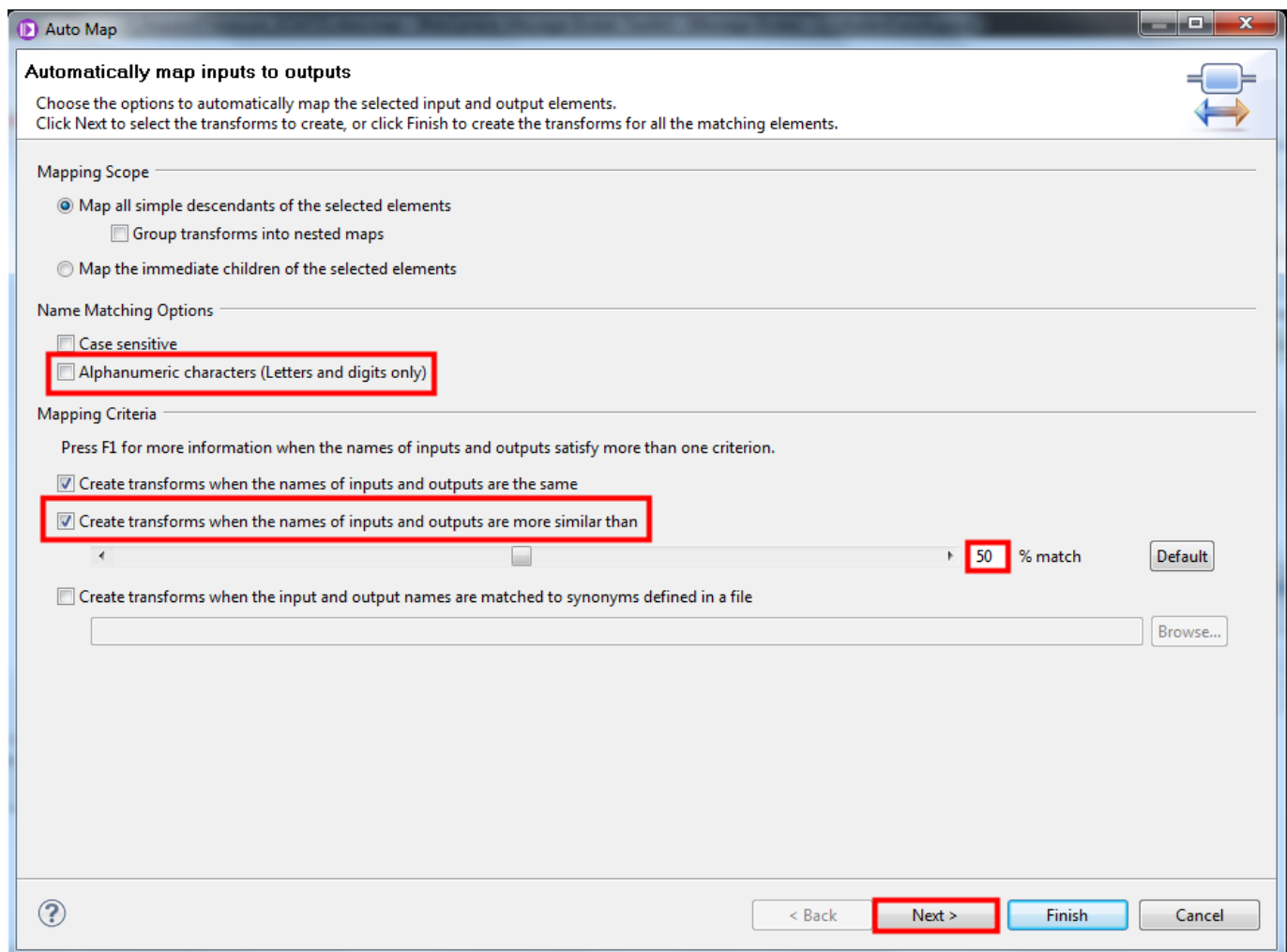


The local map will open.

__93. Select the auto map icon to start an auto map operation.



- __94. Select the check box for **Create mappings when source and target names are more similar than**.
- __95. Set the **% match** to **50** using the slider bar or by overtyping the value.
- __96. **Uncheck** the box for alphanumeric characters.
- __97. Click **Next**.

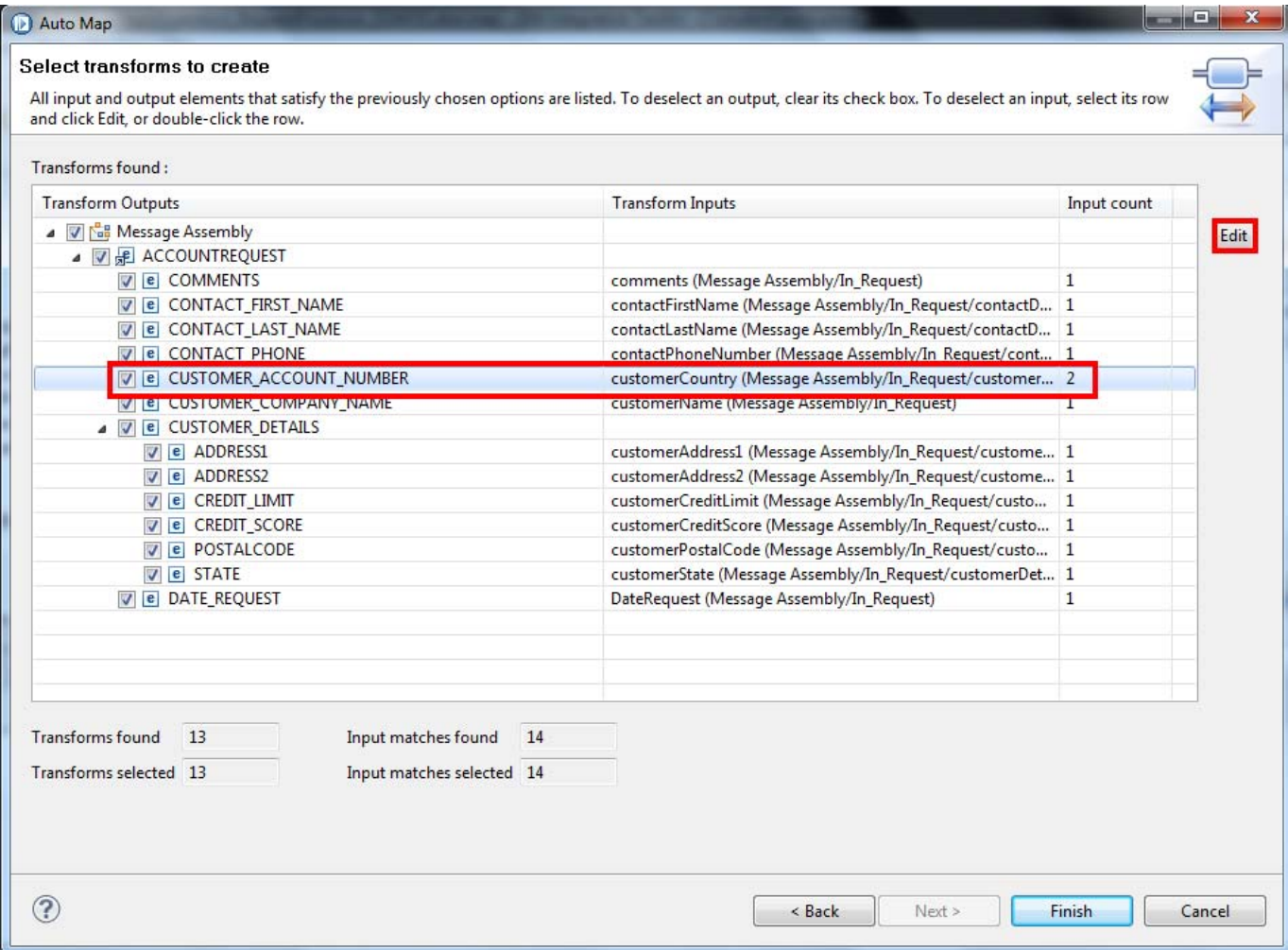


A comparison of the names will be performed and the results shown on the next screen.

In this case the **CUSTOMER_ACCOUNT_NUMBER** target has two sources. This is not right. It will now be corrected.

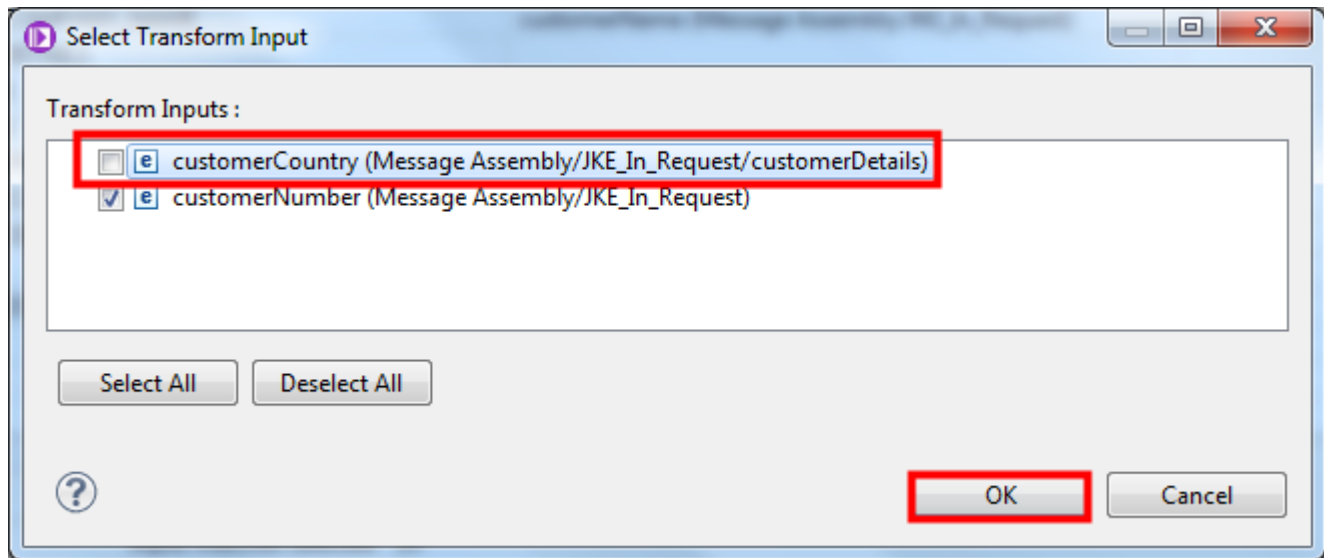
__98. **Select** the line with two sources (**CUSTOMER_ACCOUNT_NUMBER**).

__99. Click the **Edit** button.



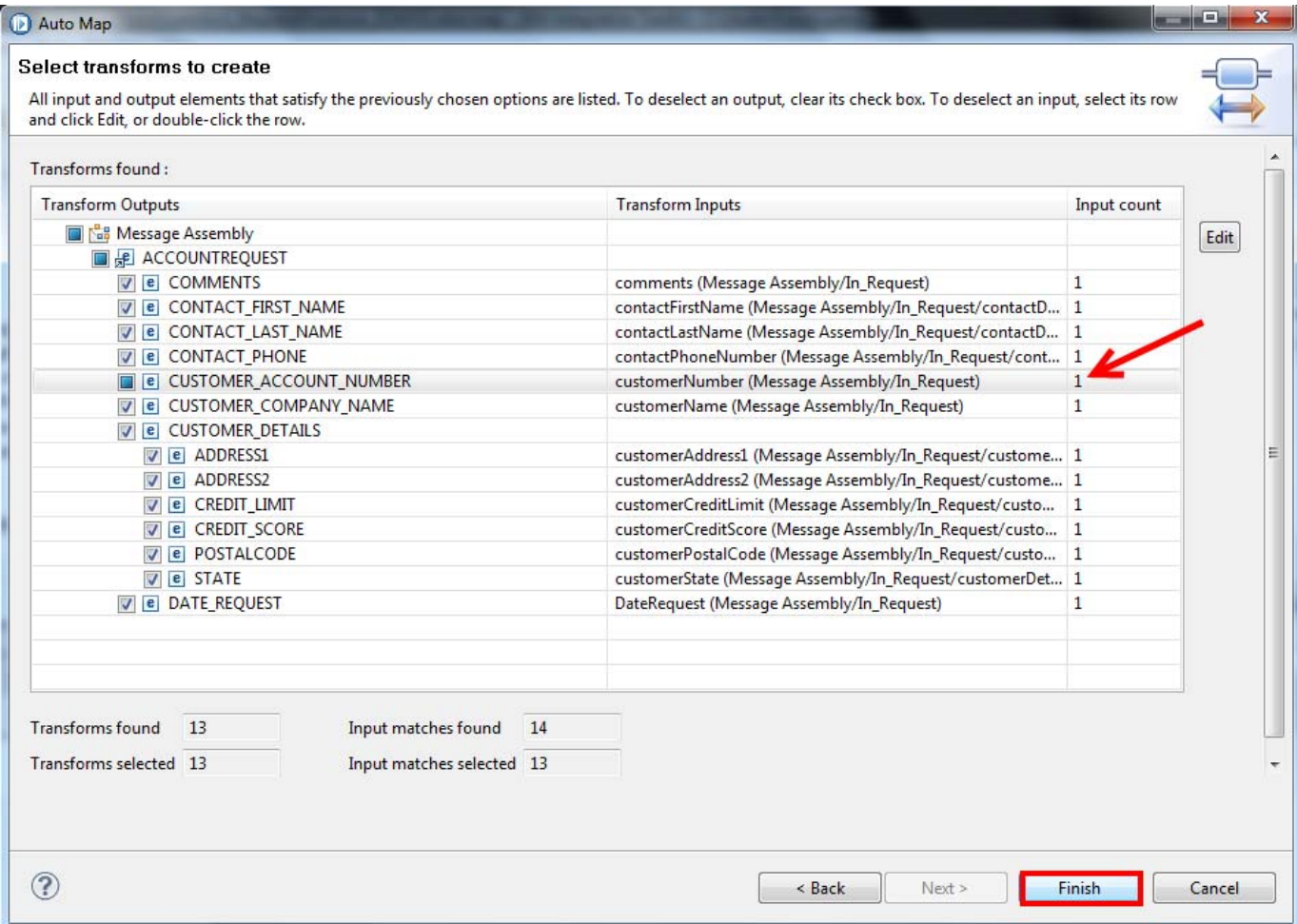
___100. Uncheck the box for **customerCountry** as this is not the proper source for this target.

___101. Click **OK** to remove the incorrect mapping.



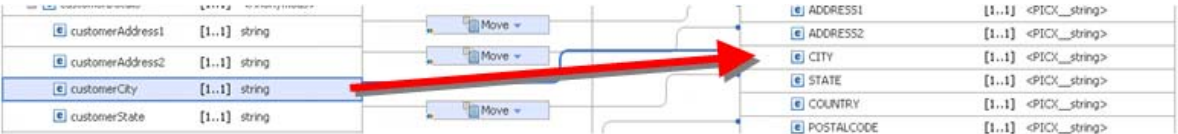
The **CUSTOMER_ACCOUNT_NUMBER** target field now has a single correct source.

__102. Click the **Finish** button to perform the auto map function.

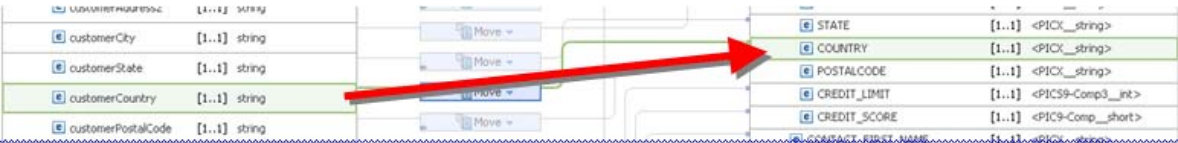


There are some required fields not mapped automatically so you need to add them now.

__103. Use drag to map the source **customerCity** field to the target **CITY** field.

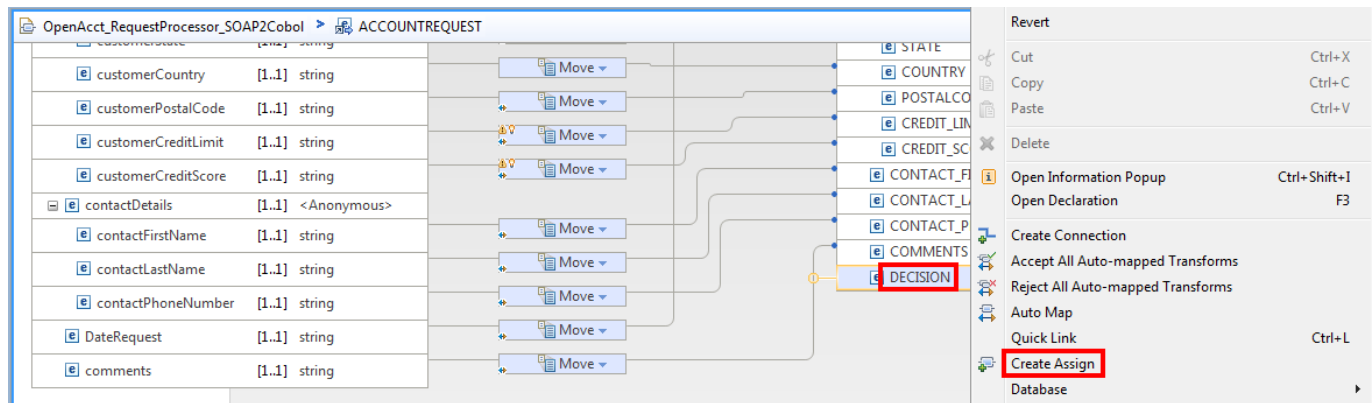


__104. Map the source **customerCountry** field dragging it onto the target **COUNTRY** field.



___105. Right-click the **DECISION** field in the target half.

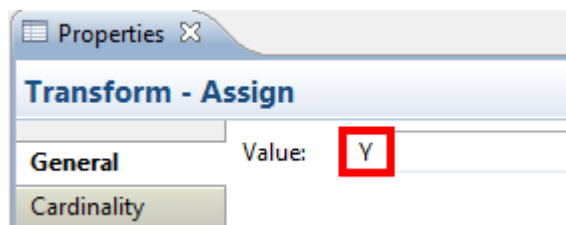
___106. Select **Create Assign** from the menu.



___107. With the new Assign map selected go to the **Properties** tab.

___108. Select the **General** tab.

___109. Enter a single character **Y** as the **Value** for this assignment. The letter should be upper case.



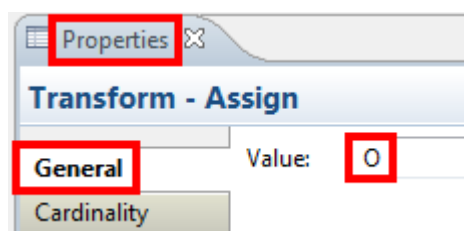
___110. Right click the **ACTION_REQUEST** field in the target half.

___111. Select **Create Assign** from the menu.

___112. With the new Assign map selected go to the **Properties** tab.

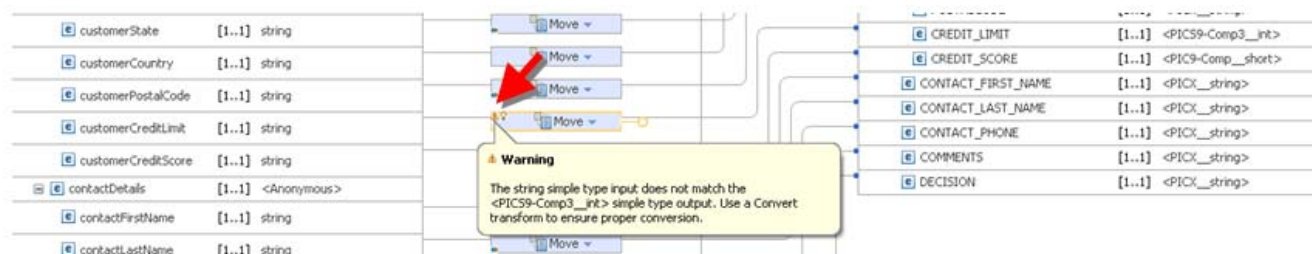
___113. In the **Properties** pane select the **General** tab.

___114. Enter a single character (letter) **O** as the value for this assignment. The letter should be upper case.



Now it is time to address the two warnings within the generated map.

- __115. Locate the **Move** map for the **customerCreditLimit**. Hover over the warning symbol (a small yellow triangle containing an exclamation mark) to reveal the details of the warning. The warning text is: **The string simple type input does not match the <PICS9-Comp3__int> simple type output. Use a Convert transform to ensure proper conversion.**



- __116. Move the cursor to hover over the **light bulb** (just next to the warning symbol).

- __117. Select the quick fix that is being offered: **Use convert to ensure the proper conversion.**



Repeat this process to resolve the remaining warning in the map.

- __118. Locate the Move map for the **customerCreditScore**.

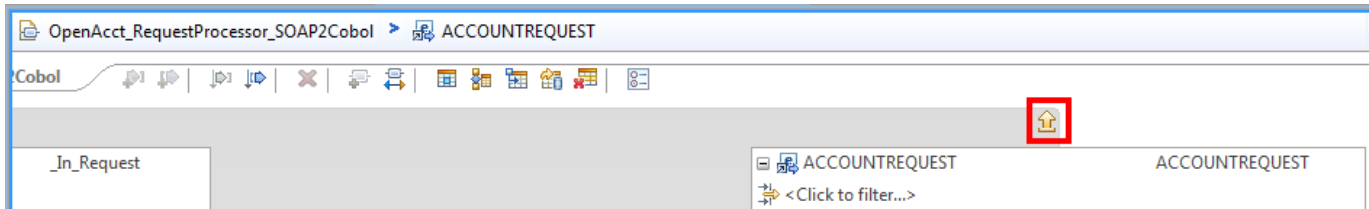
- __119. Once again move the cursor to hover over the **light bulb** (just next to the warning symbol).

- __120. Select the quick fix that is being offered: **Use convert to ensure the proper conversion**



The local map is now complete.

___121. Return to the top-level map by clicking the upward yellow arrow.

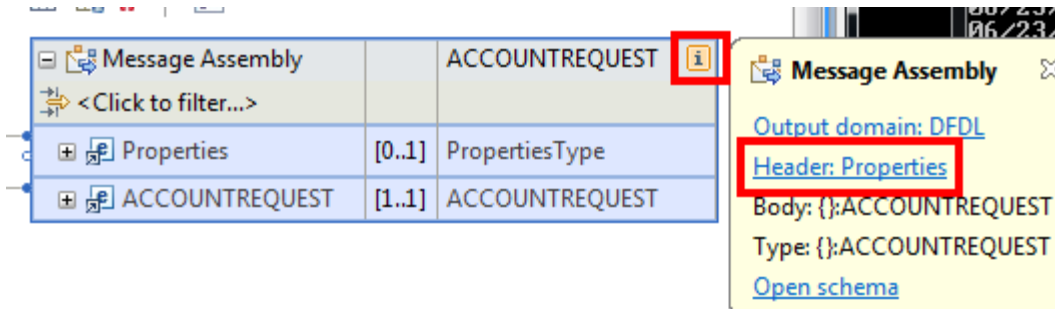


There is one more consideration for the map to be complete. The input message arrives over an HTTP transport. The output message will be sent over WebSphere MQ. The input message will contain an HTTP input header, which should be removed before sending the message to MQ. The removal of the HTTP header could also be accomplished using an HTTPHeader node. However, in this case the mapping node will be used to remove the header. This is accomplished by adding the header on the target side of the map but not mapping anything to it.

The MQMD header will then be added by the MQ Output node as this puts a message to the queue.

___122. In the output **Message Assembly** select the small blue “i” symbol to the right.

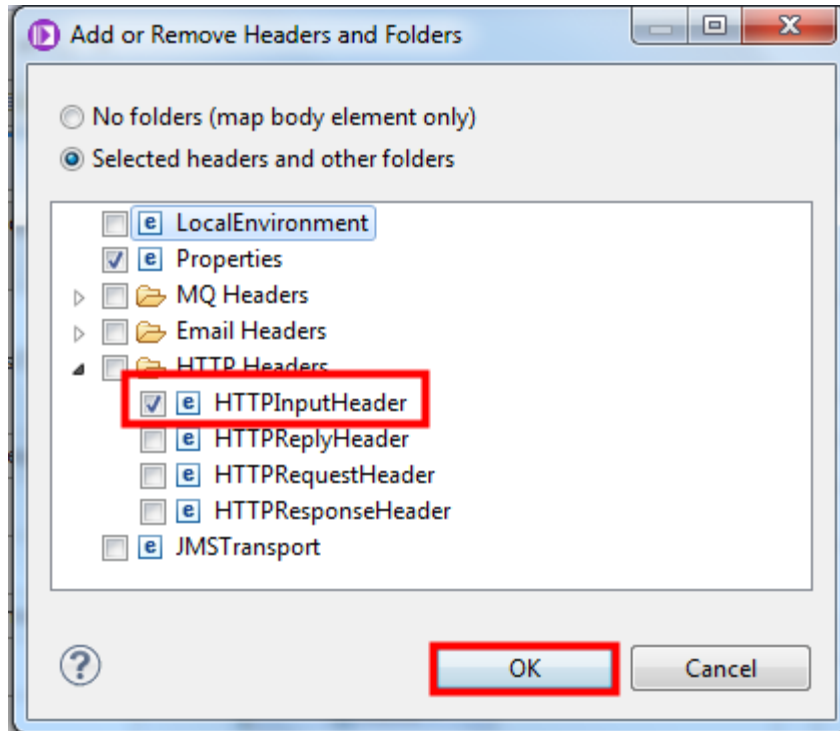
___123. Click the **Header: Properties** link.



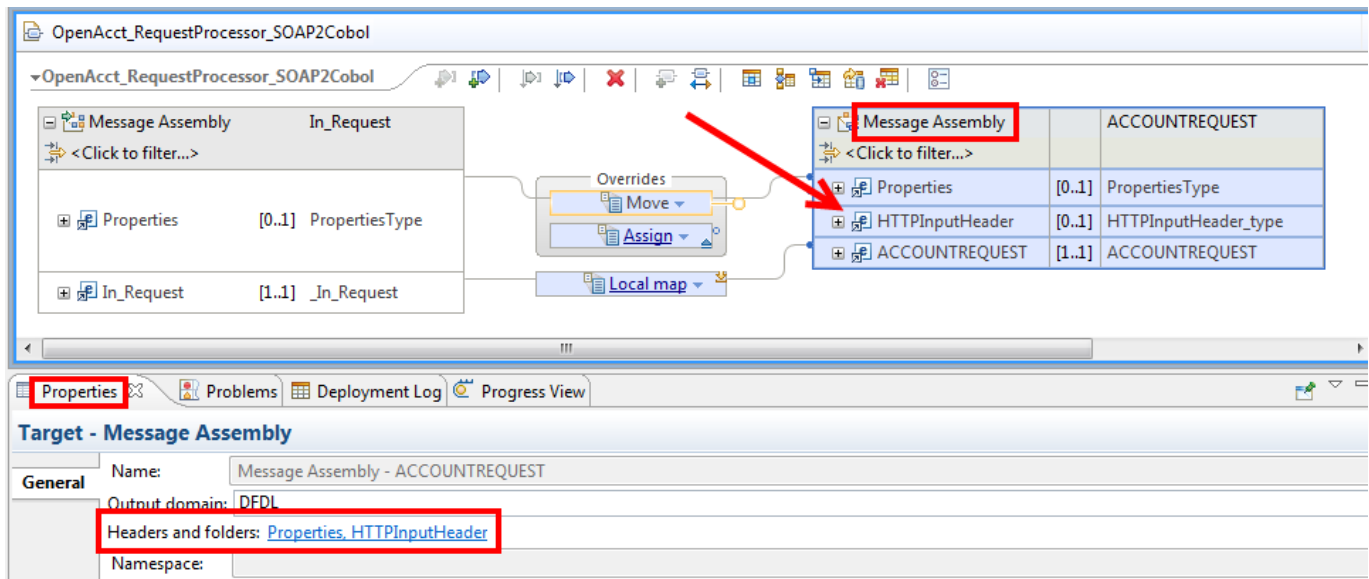
___124. Expand the **HTTP Headers** folder (Do not select all the headers at this level).

___125. Select the check box next to the **HTTPInputHeader** header.

___126. Press the **OK** button to add the header.



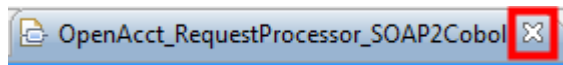
The **HTTPInputHeader** is now shown in both the graphical map (shown here with the target properties minimized) and the **Headers and folders** property. Since nothing is mapped to this header it will be deleted from the map output.



This finishes the request mapping.

__127.  Save the map (**Ctrl+S**).

__128. Close the Map editor.

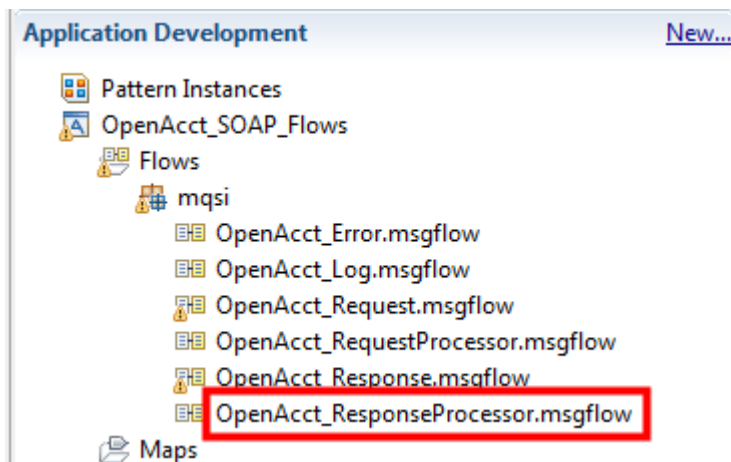


__129.  Save the **OpenAcct_RequestProcessor** message flow (**Ctrl+S**).

__130. Close the **OpenAcct_RequestProcessor** message flow editor tab.



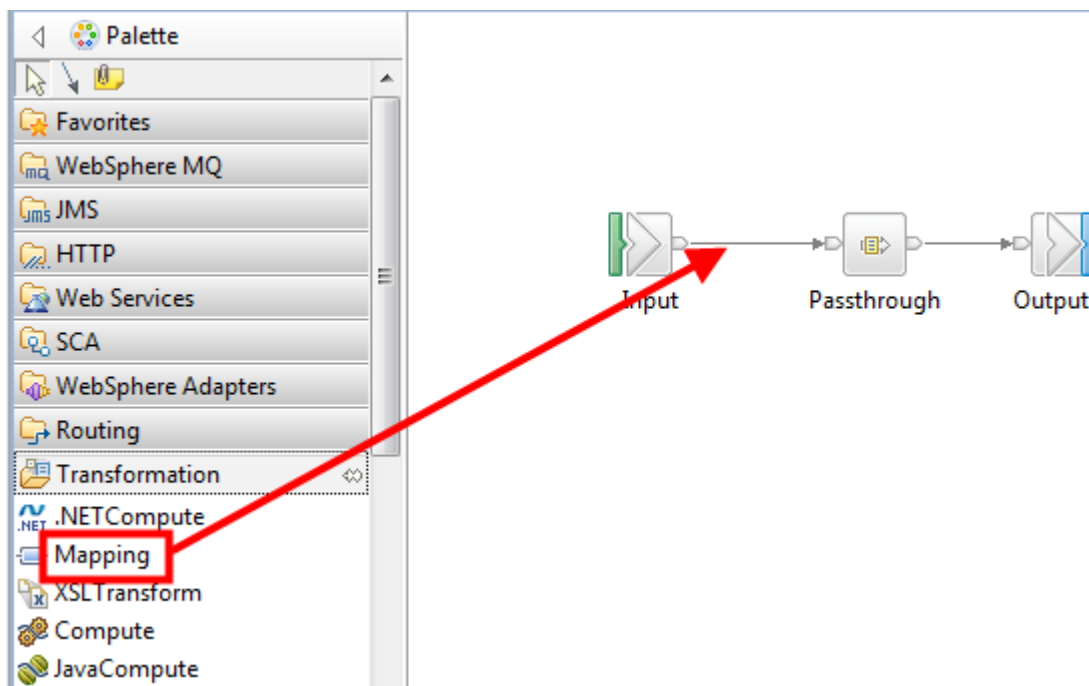
__131. Double-click the **OpenAcct_ResponseProcessor.msgflow** message flow.



__132. Expand the **Transformation** folder.

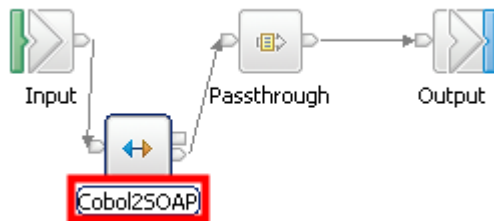
__133. Drag a Mapping node from the palette directly on the connector between the **Input node** and the **Passthrough node**.

__134. Drop the node when the connector color changes to blue.

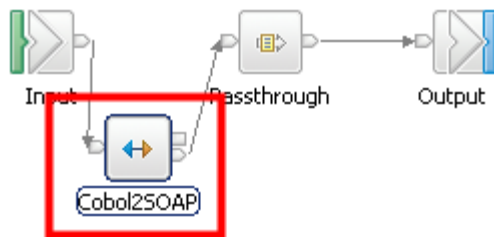


The mapping node should now be connected with both existing nodes.

__135. Change the name of the mapping node to **Cobol2SOAP**.



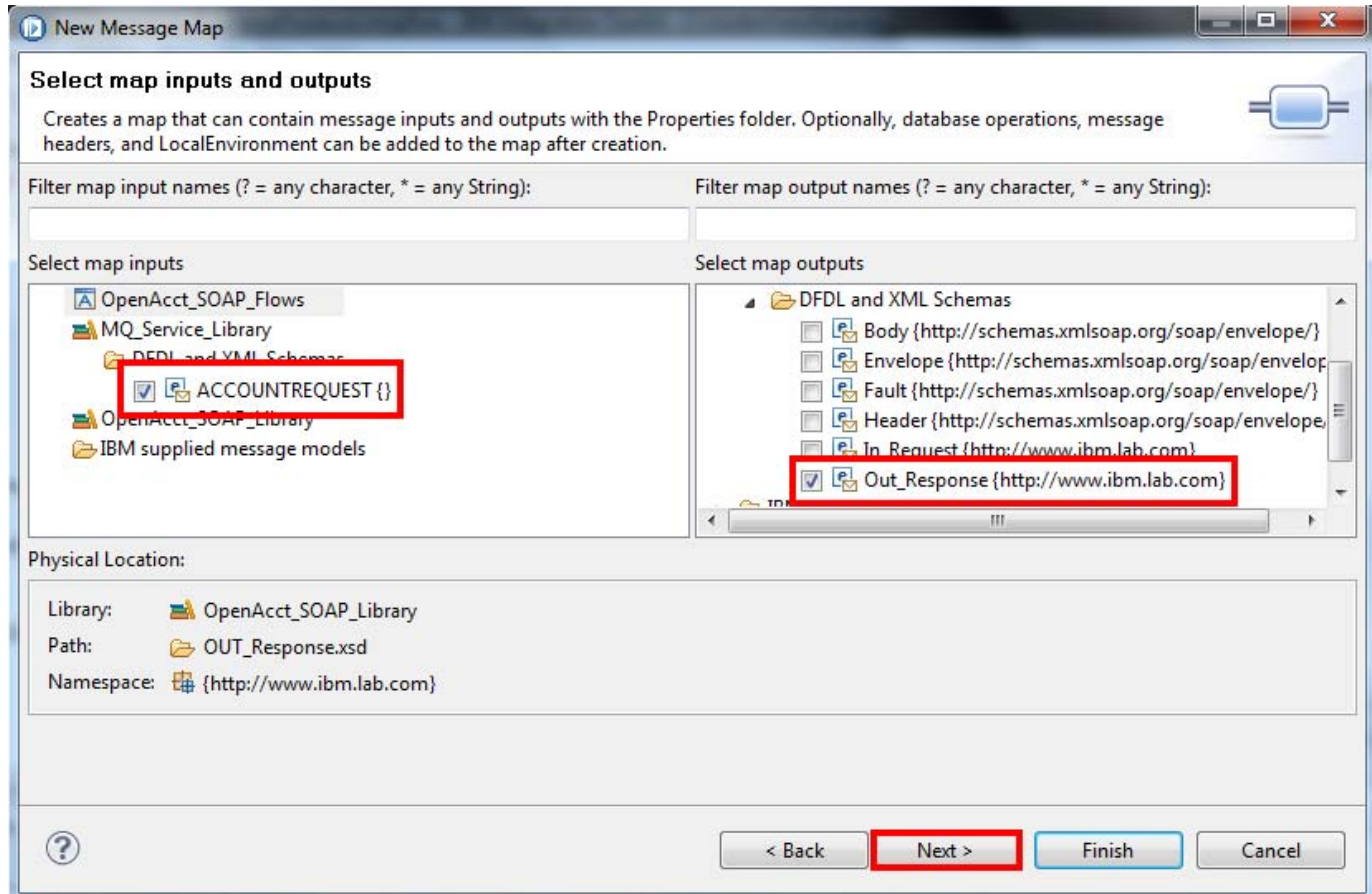
__136. Double-click the new **Cobol2SOAP** mapping node



__137. Accept the default values for the map name and location.

__138. Press the **Next** button to continue.

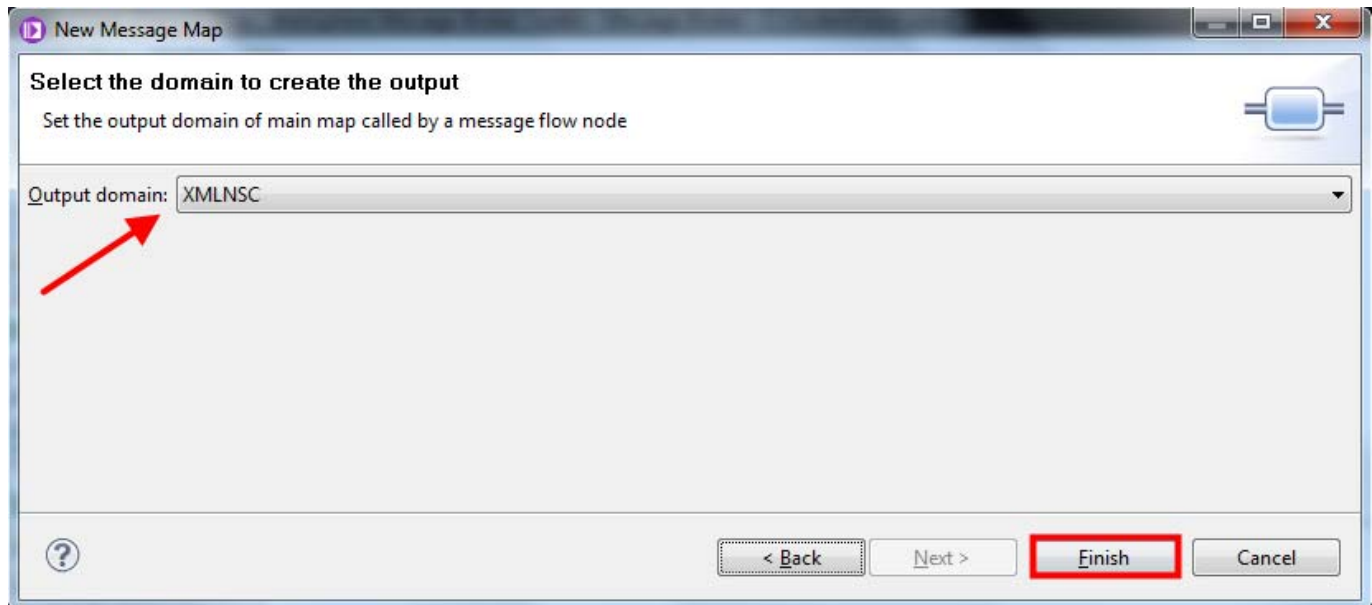
- ___139. In the left hand pane, expand the **MQ_Service_Library**→**DFDL and XML Schemas** folders.
- ___140. Check the box next to **ACCOUNTREQUEST**. This will be the input message for the mapping.
- ___141. In the right hand pane, expand the **OpenAcct_SOAP_Library**→**DFDL and XML Schemas** folders.
- ___142. Check the box next to **Out_Response**. This will be the output message for the mapping.
- ___143. Click **Next**.



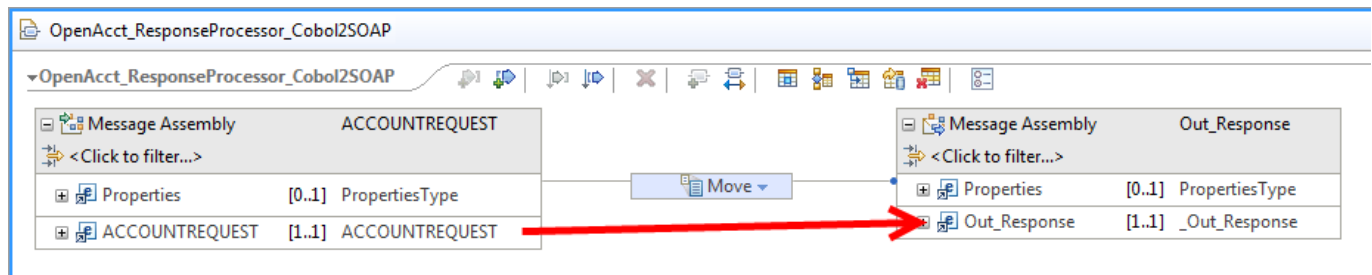
The output message will be the reply to the web service in SOAP XML.

__144. Leave **XMLNSC** selected as the **Output domain**.

__145. Click **Finish**.

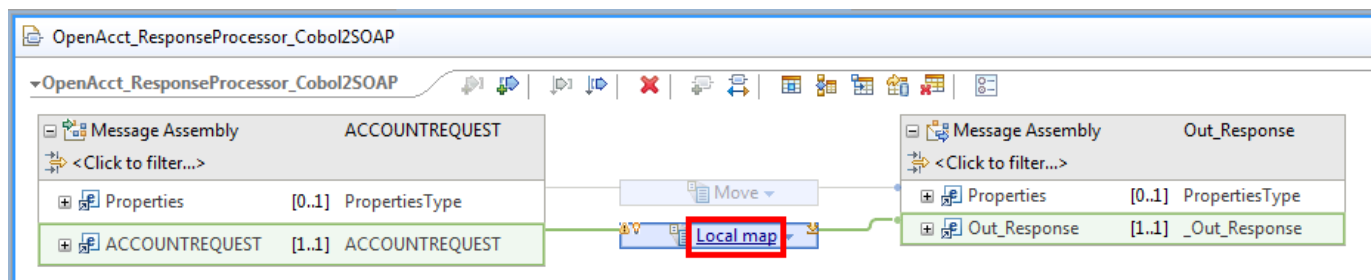


__146. Map the **ACCOUNTREQUEST** input to the **Out_Response** output.



A local mapping operation will be created.

__147. Click the **Local map** operation.



Expand the source and target message bodies.

__148. Create assignments by dragging:

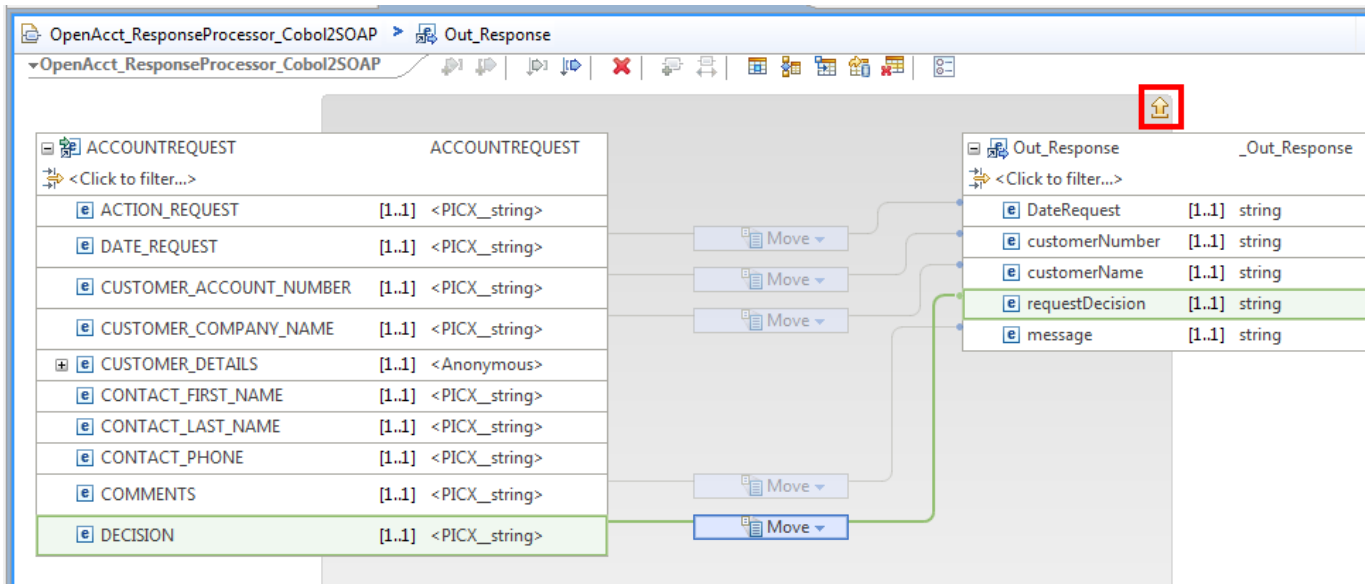
- __a. **DATE_REQUEST** to **DateRequest**
- __b. **CUSTOMER_ACCOUNT_NUMBER** to **customerNumber**
- __c. **CUSTOMER_COMPANY_NAME** to **customerName**
- __d. **DECISION** to **requestDecision**
- __e. **COMMENTS** to **message**

The screenshot displays the IBM Integration Bus v9 interface for mapping message bodies. The left pane shows the source message body, **ACCOUNTREQUEST**, and the right pane shows the target message body, **Out_Response**. Red arrows indicate the following mappings:

Source Field	Target Field
DATE_REQUEST	DateRequest
CUSTOMER_ACCOUNT_NUMBER	customerNumber
CUSTOMER_COMPANY_NAME	customerName
DECISION	requestDecision
COMMENTS	message

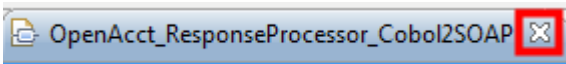
This finishes the response mapping.

__149. Click the yellow up arrow to return to the top level map.



__150.  Save the map (**Ctrl+S**).

__151. Close the Map editor.



__152.  Save the **OpenAcct_ResponseProcessor** message flow (**Ctrl+S**).

__153. Close the **OpenAcct_ResponseProcessor** message flow.

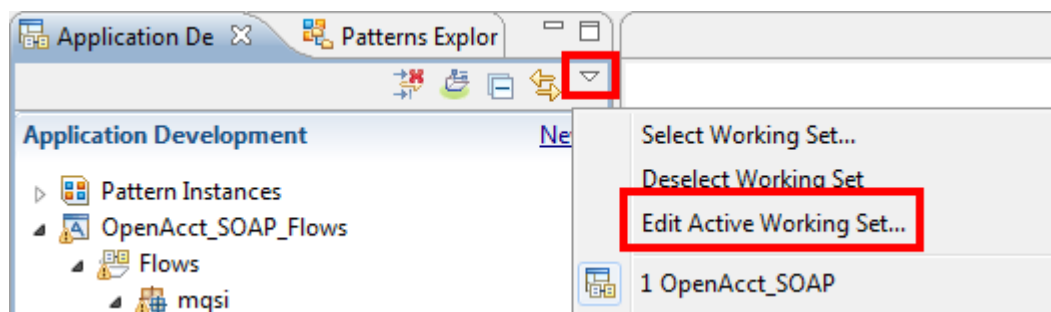


4.2 Test the message flow

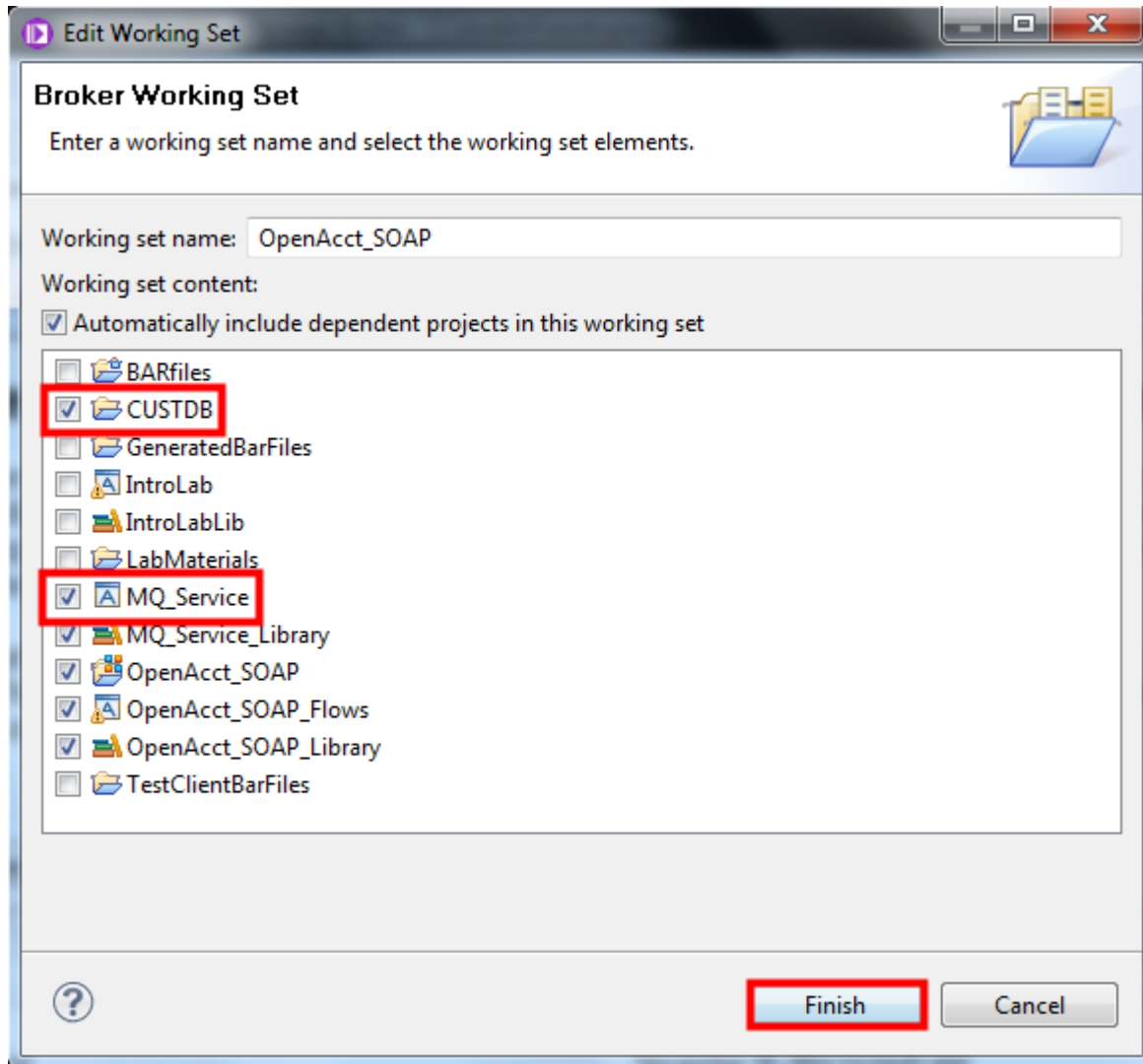
The Test Client can be used to test flows that begin with MQInput, HTTPInput, JMSInput, SCAInput or SOAPInput nodes. The Test Client will automate much of that process for us. It will create and populate a BAR file, deploy the BAR file to the integration server, submit a test message, monitor any output nodes for results and display the results.

The message flow depends on a message flow in the MQ_Service application. This message flow simulates a back end account open service based on MQ. This application will be deployed first.

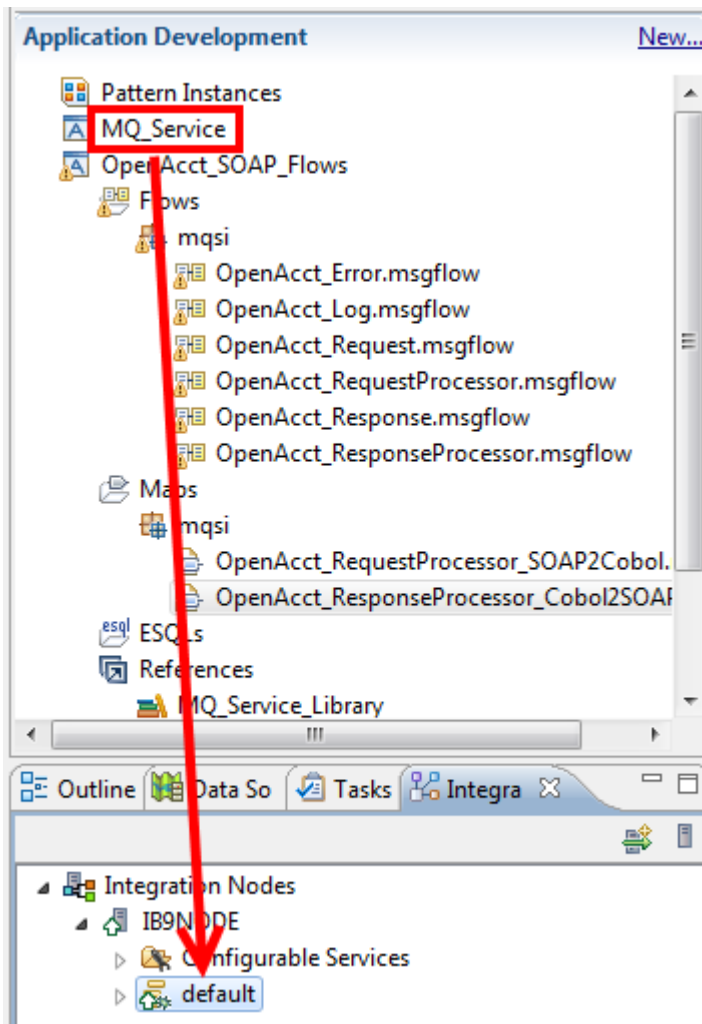
- __1. Select the small triangle on the right near the top of the project navigator.
- __2. Select **Edit Active Working Set** from the menu.



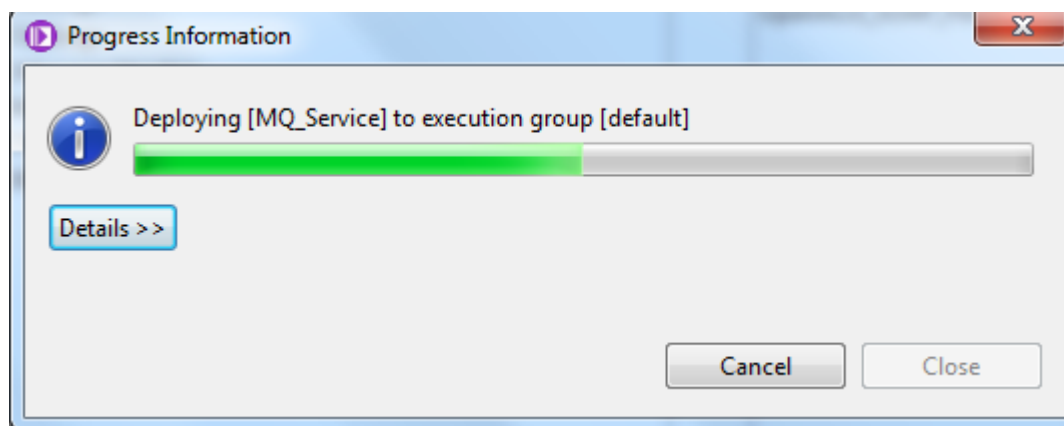
- __3. Select the check box next to the **CUSTDB** project.
- __4. Select the check box next to the **MQ_Service** project.
- __5. Press the **Finish** button.



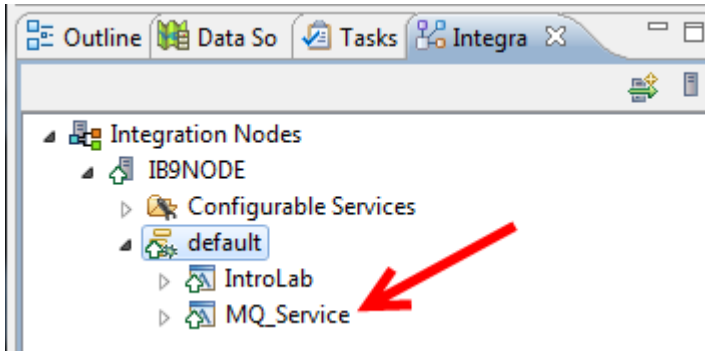
- __6. Drag the **MQ_Service** application to the **default** integration server.



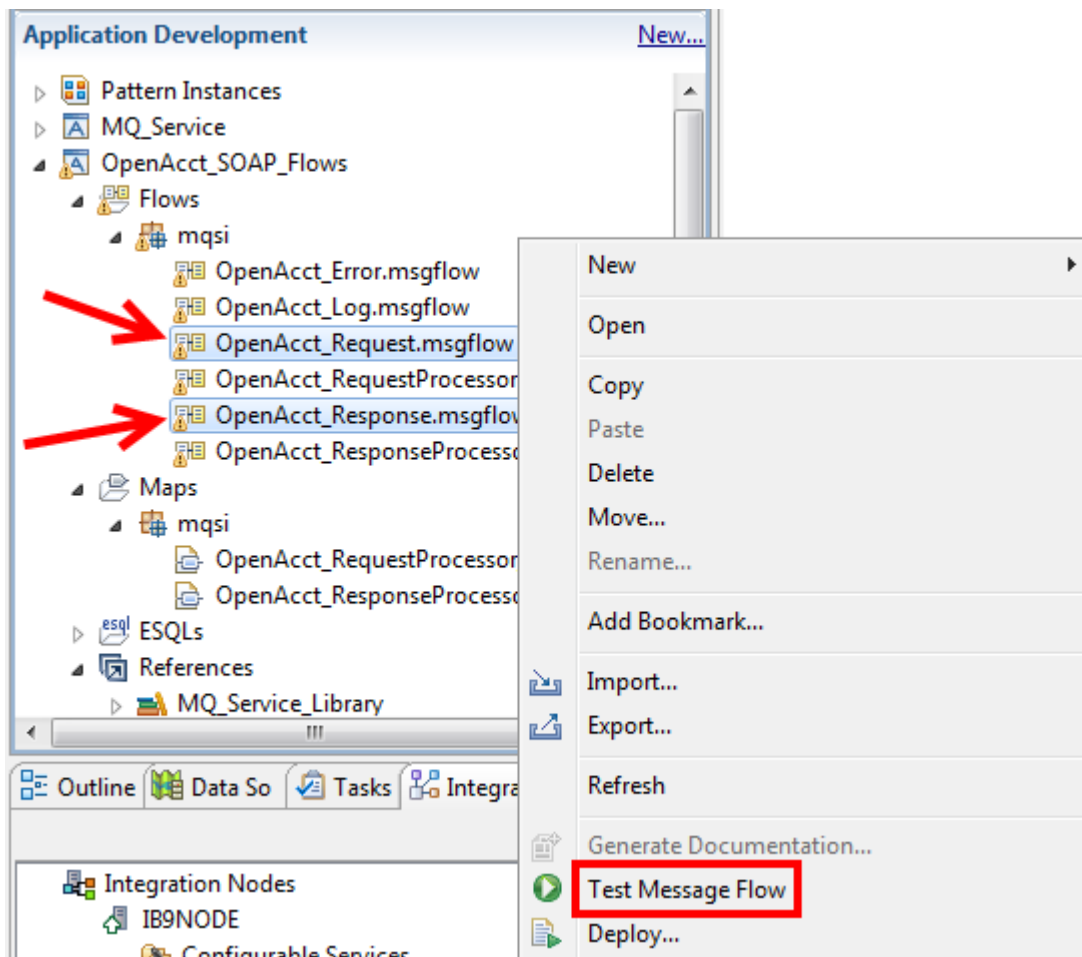
A deployment operation will start.



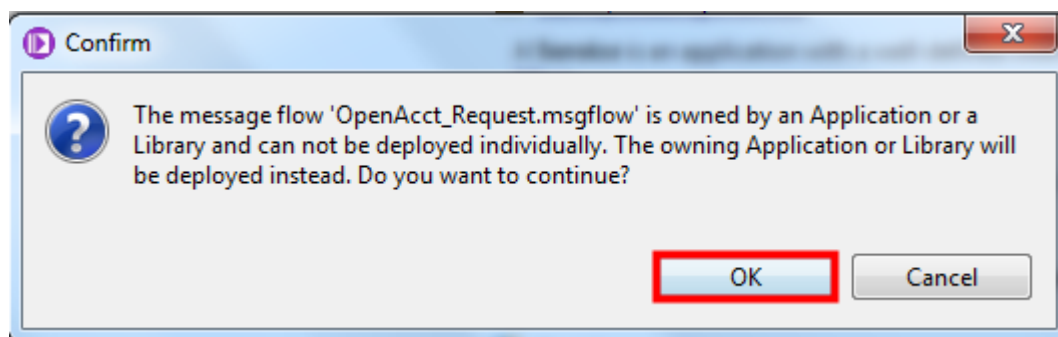
The **MQ_Service** application should be visible under the **default** integration server.



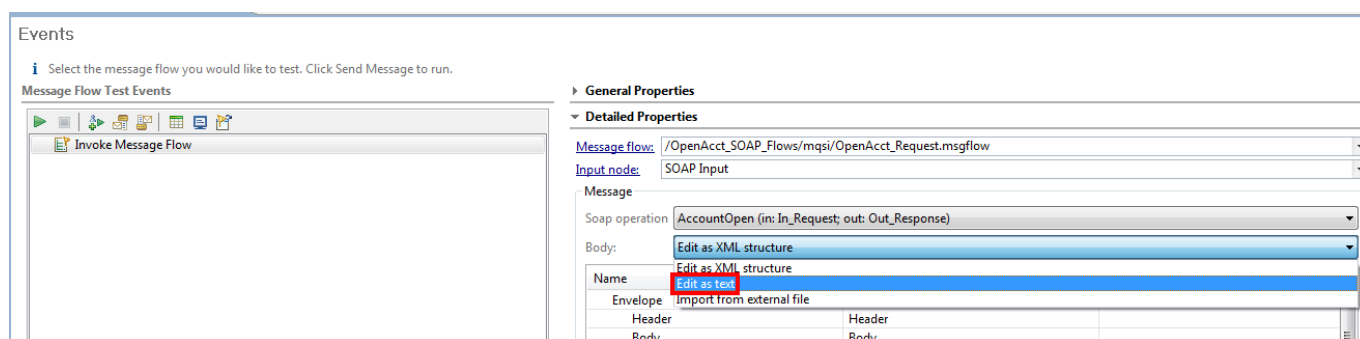
- __7. Select the **OpenAcct_Request** message flow.
- __8. Hold down the **Ctrl** key and select the **OpenAcct_Response** message flow.
- __9. Press the right mouse button.
- __10. Select **Test Message Flow** from the context menu. This starts the Test Client.



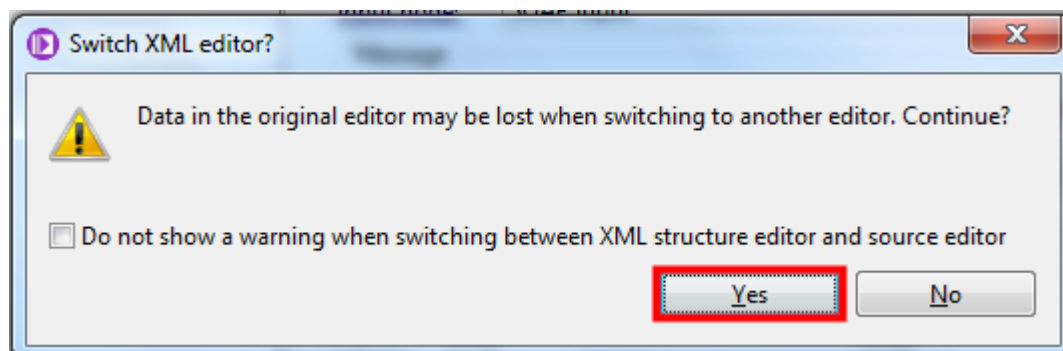
__11. Press the **OK** button to dismiss the warning.



__12. Use the **Body** pull-down menu to select the **Edit as text** option.



__13. If a warning is displayed press the **Yes** button. This warning will not appear if the check box was selected in an earlier lab.



___14. Click the **Import Source** button.

Message flow: /OpenAcct_SOAP_Flows/mqsi/OpenAcct_Request.msgflow

Input node: SOAP Input

Message

Soap operation: AccountOpen (in: In_Request; out: Out_Response)

Body: Edit as text

```
<?xml version="1.0" encoding="UTF-8"?><tns0:Envelope xmlns:tns0="http://schemas.xml
<tns0:Header/>
<tns0:Body>
  <tns1:In_Request>
    <customerNumber>customerNumber</customerNumber>
    <customerName>customerName</customerName>
    <customerDetails>
      <customerAddress1>customerAddress1</customerAddress1>
      <customerAddress2>customerAddress2</customerAddress2>
      <customerCity>customerCity</customerCity>
      <customerState>customerState</customerState>
      <customerCountry>customerCountry</customerCountry>
      <customerPostalCode>customerPostalCode</customerPostalCode>
```

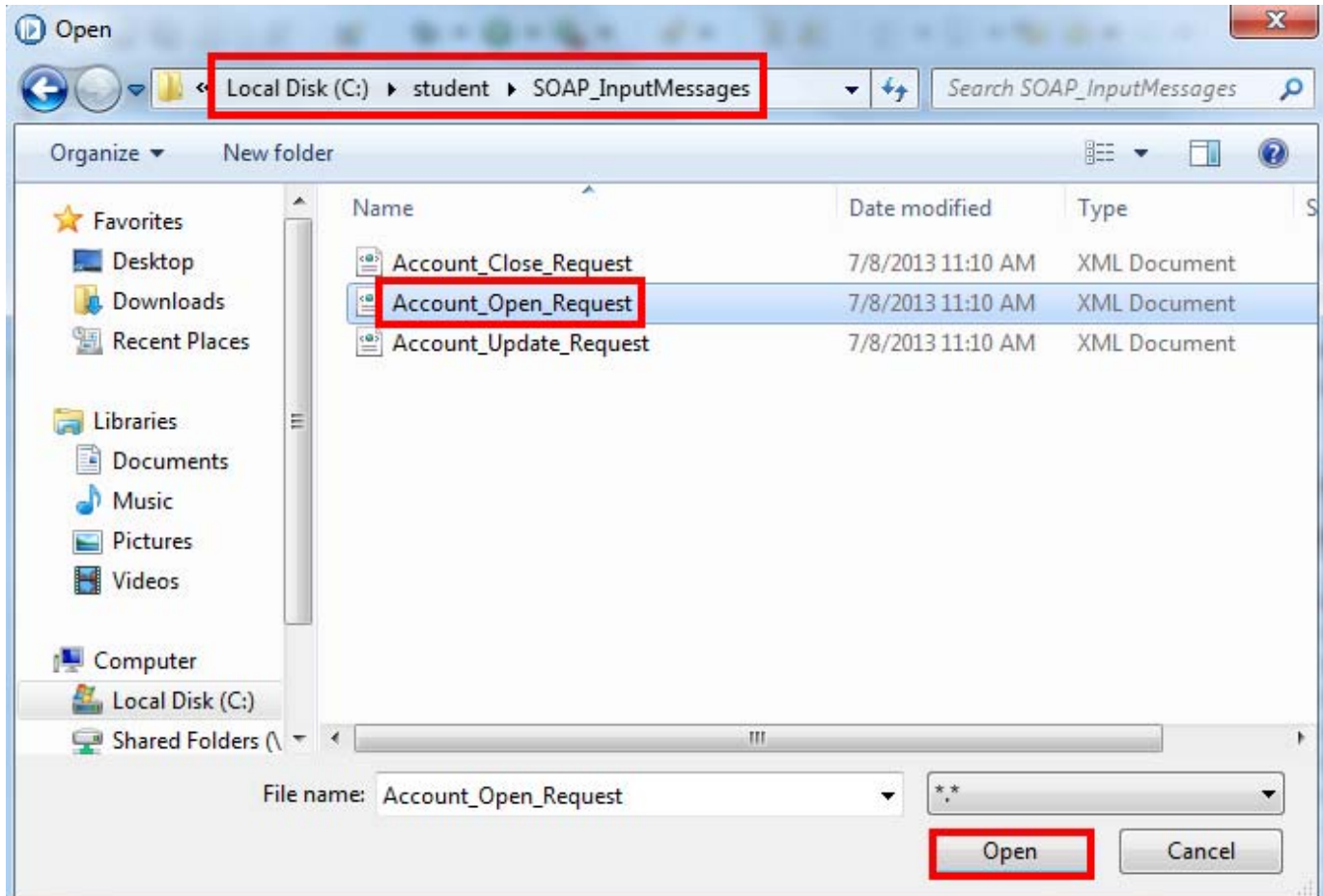
Import Source...

Send Message

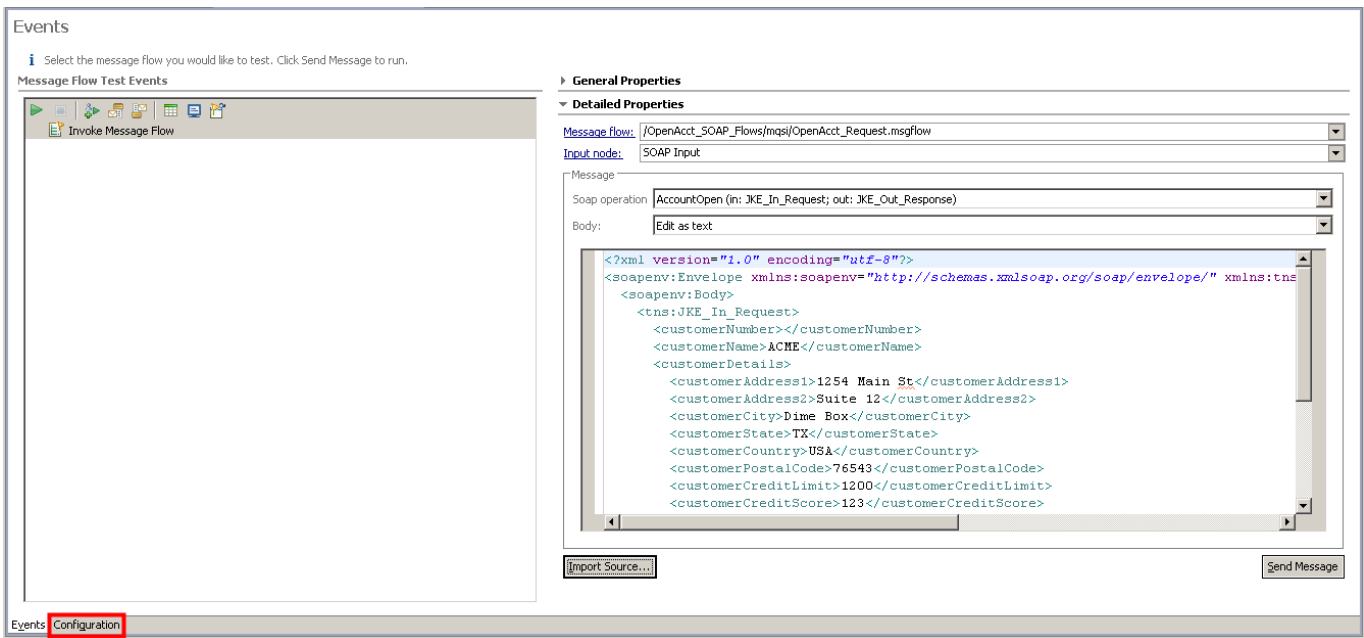
__15. Navigate to the **C:\student\SOAP_InputMessages** directory.

__16. Select the **Account_Open_Request.xml** file.

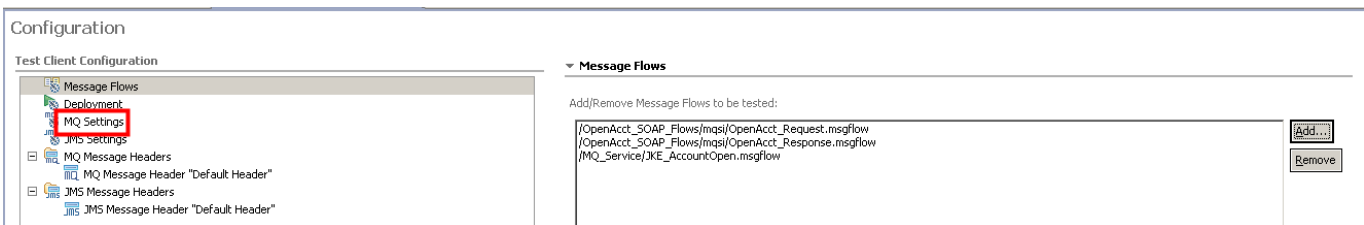
__17. Click the **Open** button.



__18. Click the **Configuration** tab.

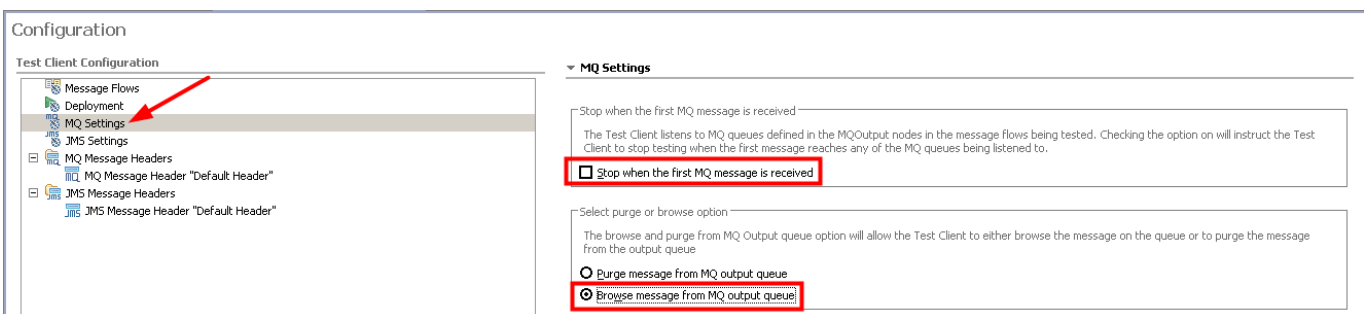


__19. Select **MQ Settings**.

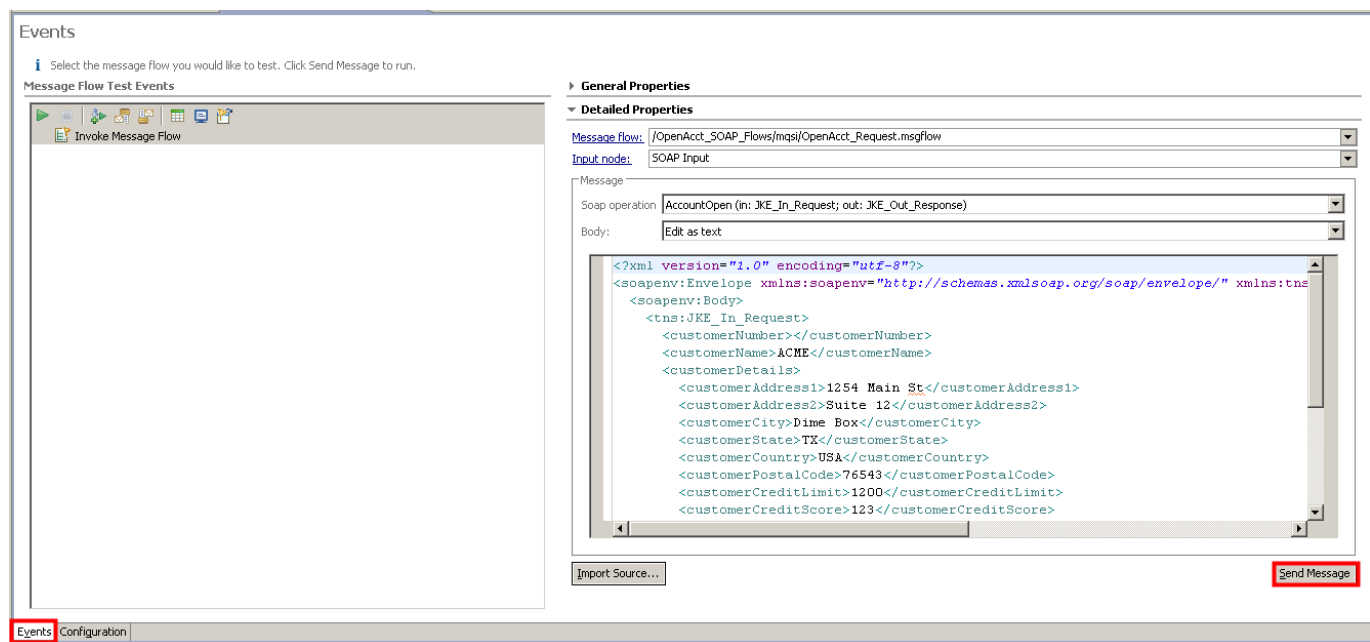


__20. Remove the selection from the **Stop when the first MQ message is received** check box.

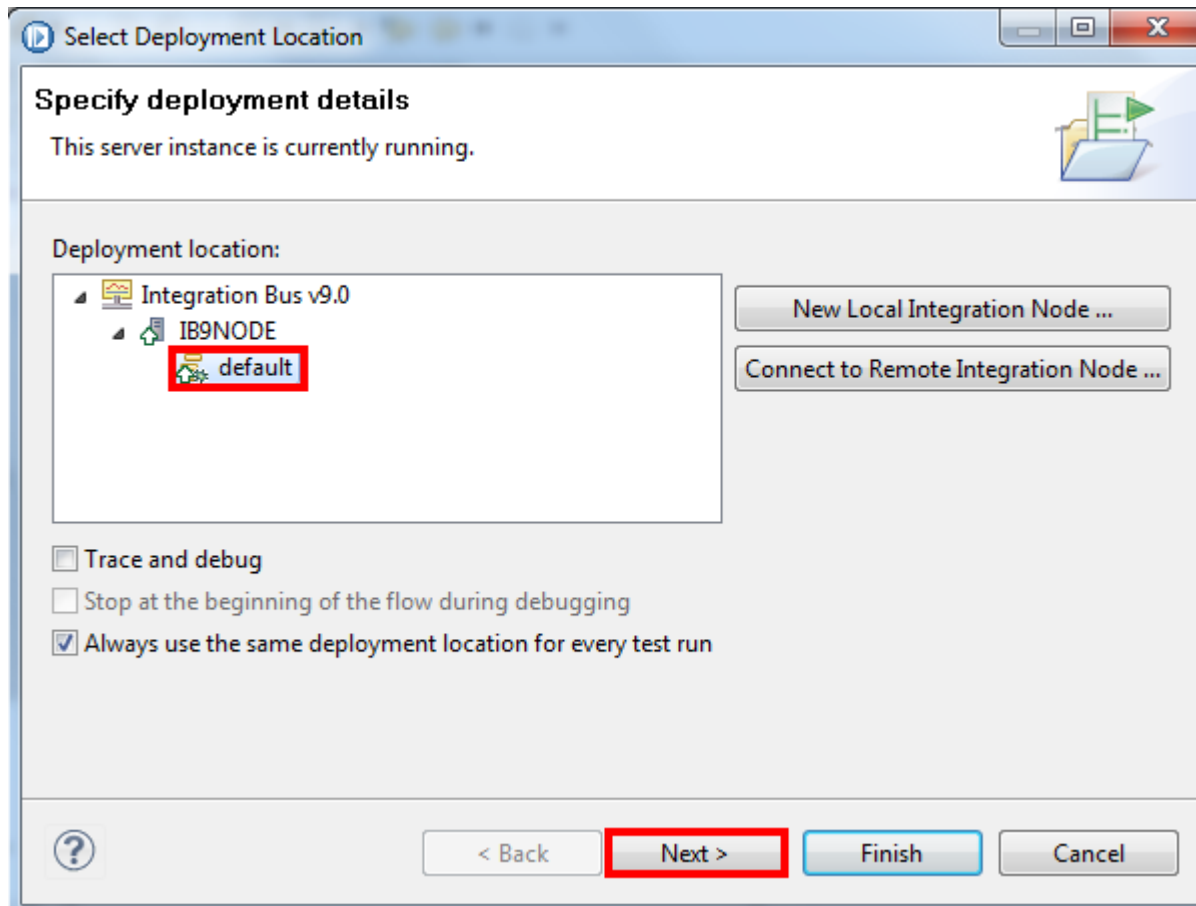
__21. Change the **Select purge or browse option** to 'Browse message from MQ output queue'.



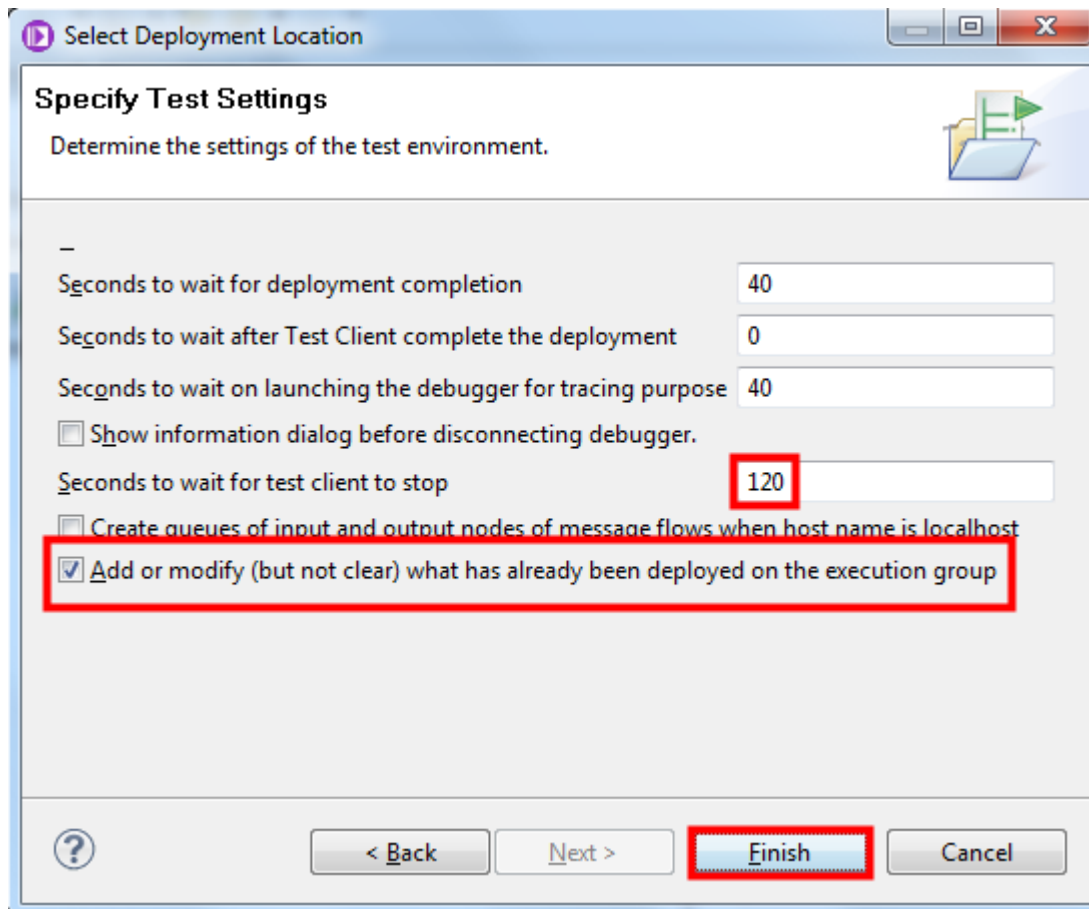
- ___22. Switch back to the **Events** tab.
- ___23. Press the **Send Message** button to initiate the test.



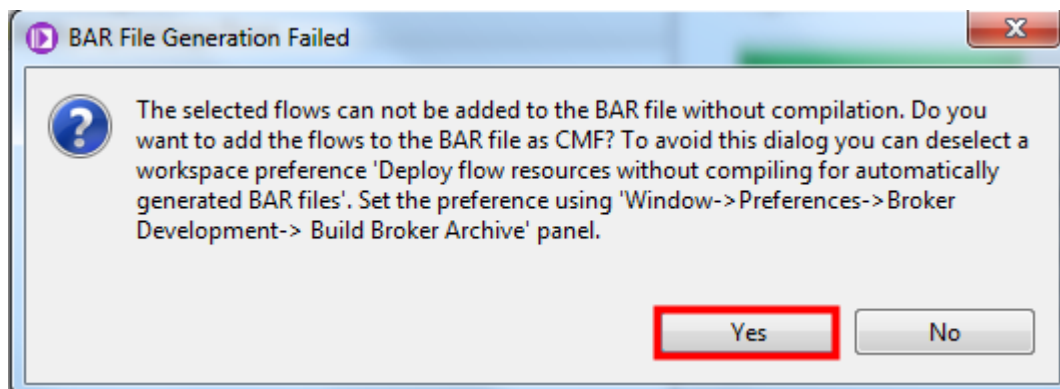
- __24. Highlight the **default** integration server.
- __25. Click **Next** (this time only).



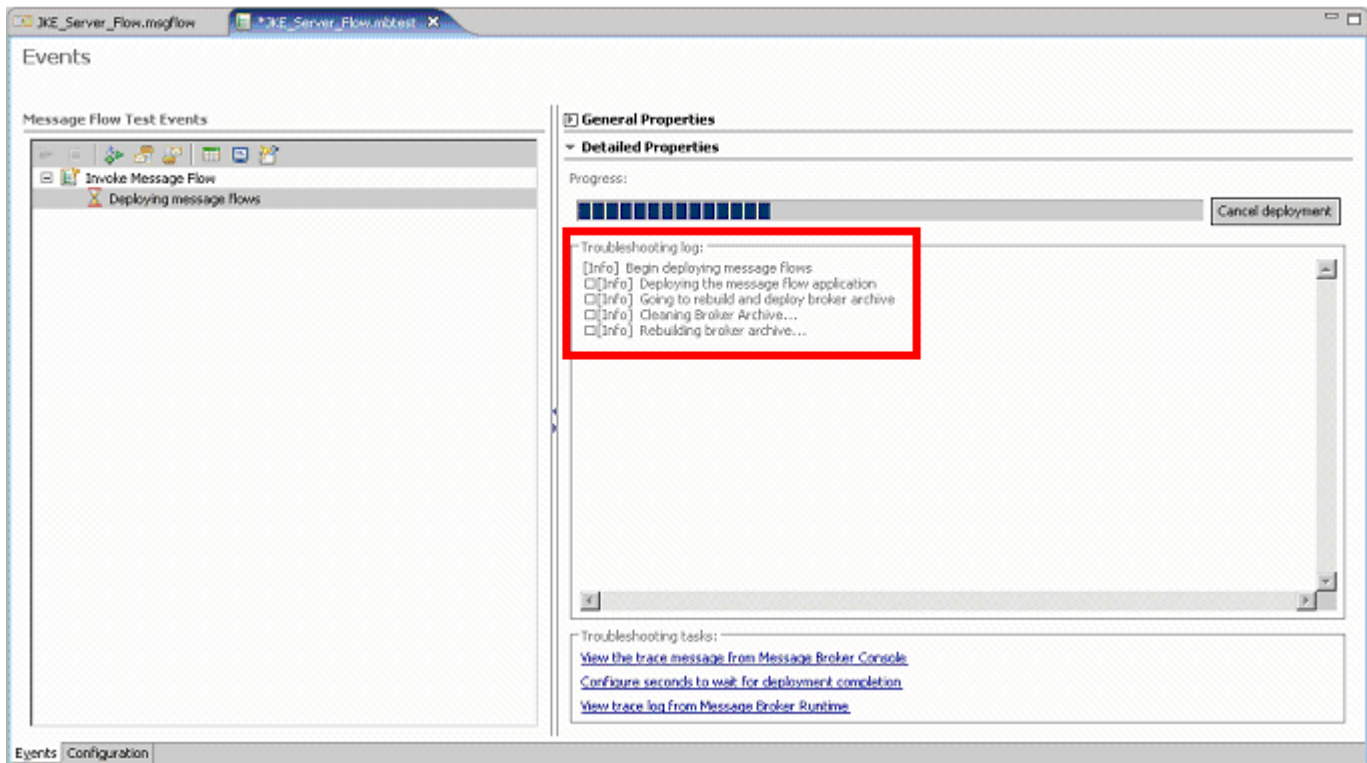
- __26. Verify that you have a value of **40** as the **Seconds to wait for deployment completion** and **Seconds to wait on launching the debugger for tracing purpose** fields.
- __27. Enter **120** as the **Seconds to wait for test client to stop**.
- __28. Make sure the **Add or modify ...** check box is checked.
- __29. Click **Finish**.



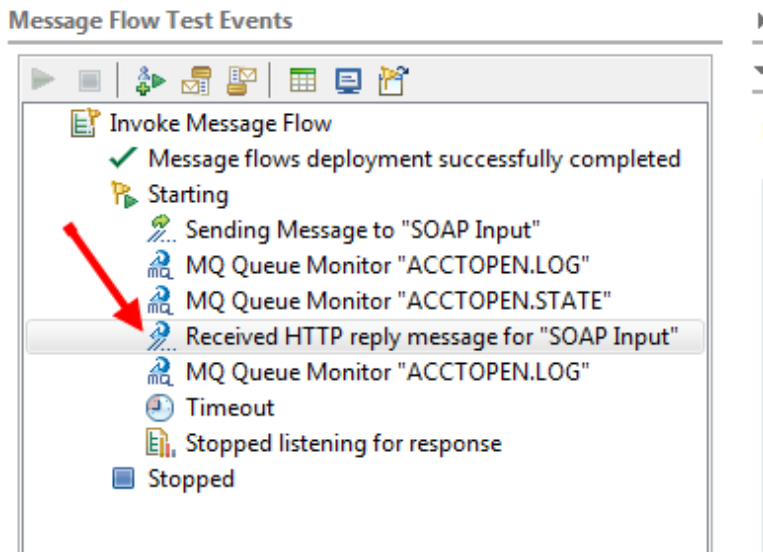
- __30. Press the **Yes** button to continue.



The Test Client should start. A progress screen shows the test client deploying the message flows and associated artifacts. The progress of these actions is detailed in the Troubleshooting log.



__31. When the response is received, it is displayed.



__32. Select the **Received HTTP reply message for "SOAP Input"** line.

Message Flow Test Events

- Invoke Message Flow
 - Message flows deployment successfully completed
 - Starting
 - Sending Message to "SOAP Input"
 - MQ Queue Monitor "ACCTOPEN.LOG"
 - MQ Queue Monitor "ACCTOPEN.STATE"
 - Received HTTP reply message for "SOAP Input"**
 - MQ Queue Monitor "ACCTOPEN.LOG"
 - Timeout
 - Stopped listening for response
 - Stopped

Detailed Properties

Endpoint URL: `http://localhost:7800/Server`

Message

Body: **View as XML structure**

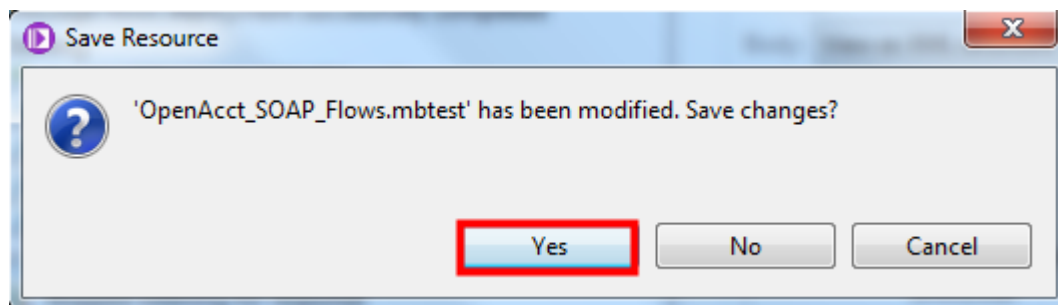
Name	Value
NS1:Envelope	
xmlns:NS1	<code>http://schemas.xmlsoap.org/soap/envelope/</code>
NS1:Body	
out:Out_Response	
xmlns:out	<code>http://www.ibm.lab.com</code>
DateRequest	10/12/2007
customerNumber	1
customerName	ACME
requestDecision	Y
message	Customer created

__33. If necessary use the small blue box icon to stop the test.

__34. Close the Test Client.



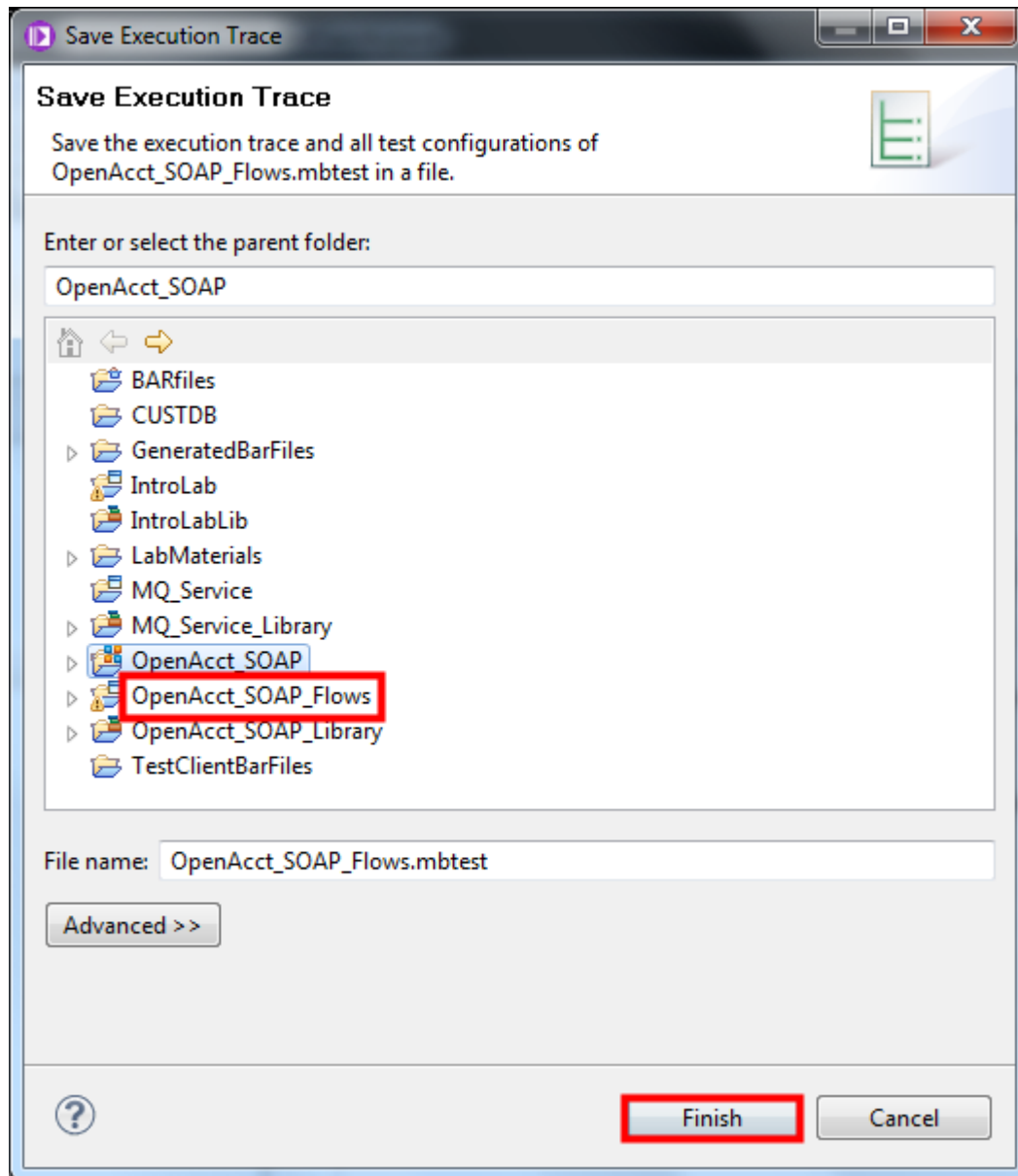
__35. Select **Yes** to save the test. The same test can be used again later without needing to rebuild it.



The ability to save multiple configurations and messages can be used to build a “test suite.” This could be used for regression testing in the future, for example.

__36. Highlight the **OpenAcct_SOAP_Flows** project.

__37. Click **Finish** to save the Test Client configuration.



This is the end of Lab 4.

Lab 5 Using .NET in IBM Integration Bus

5.1 Overview

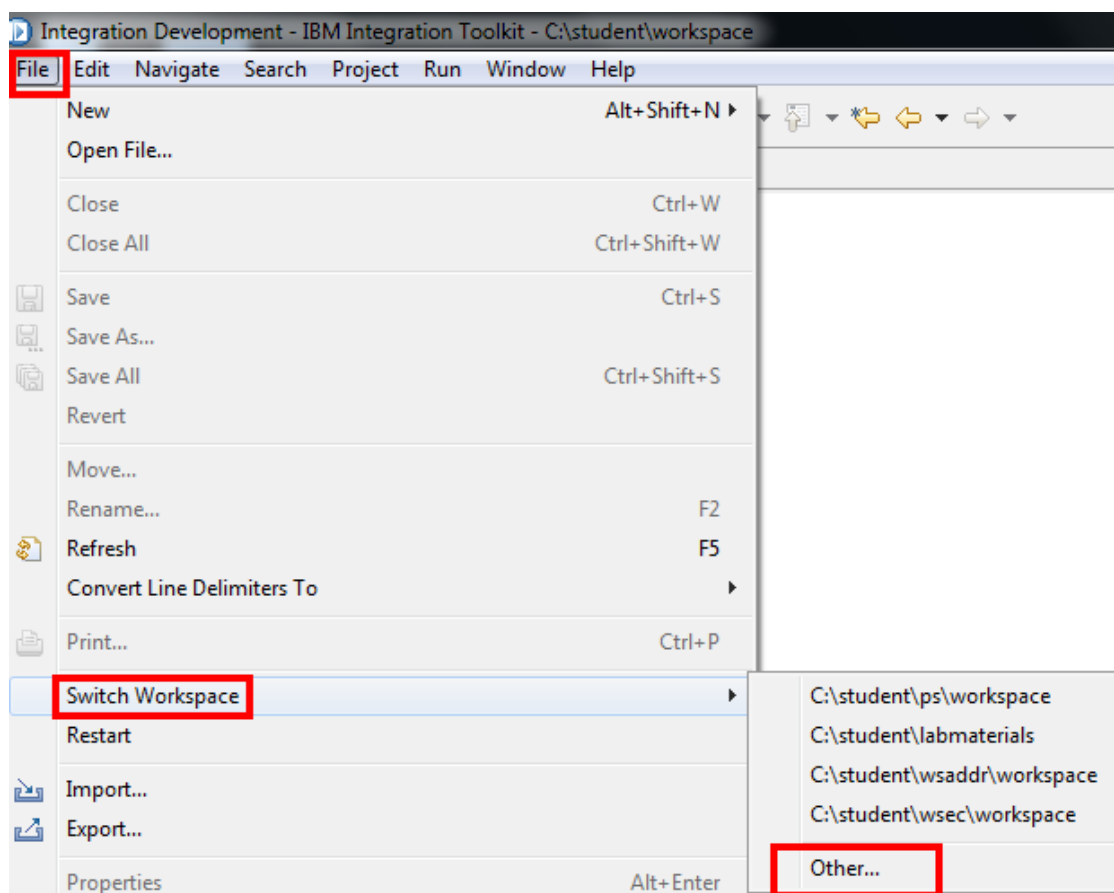
This lab will provide an introduction to using the .NET compute node in IBM Integration Bus Version 9. This lab will use the .NET compute node sample. The sample filters, modifies, and transforms messages using code written in C#. The .NET compute node can be used to construct output messages and interact with the Microsoft .NET framework (.NET) or Component Object Model (COM) applications.

IBM Integration Bus enables the hosting and execution of .NET code inside of an integration server. The .NET compute node routes or transforms messages using any Common Language Runtime (CLR) compliant .NET programming language, including C#, Visual Basic (VB), F# or C++/Common Language Infrastructure (CLI). The implementation uses V4.5 of the Microsoft .NET Framework.

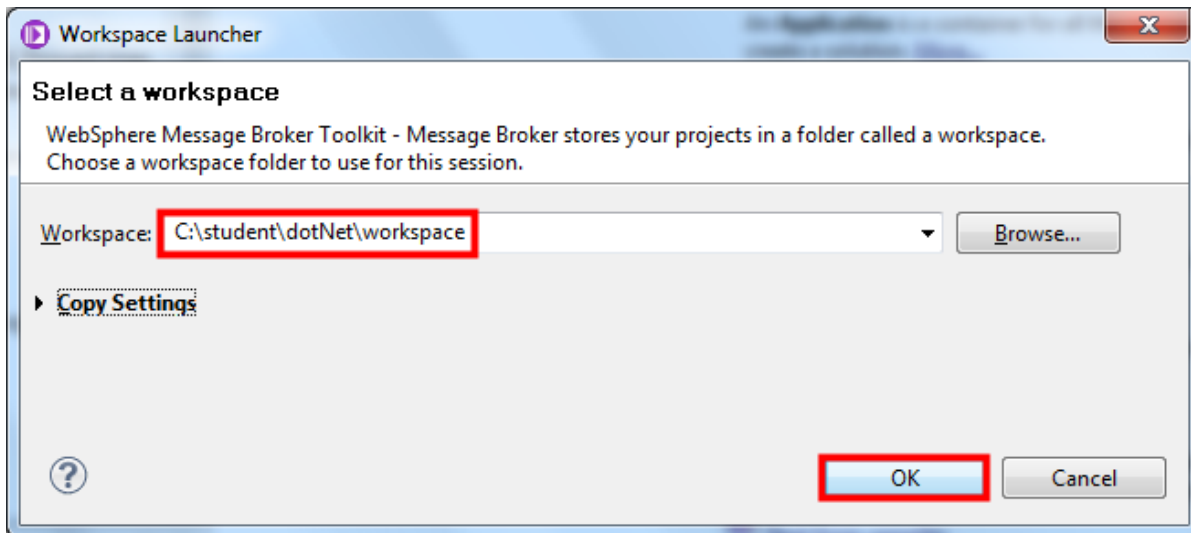
Microsoft Visual Studio Express edition is installed; if another version of Microsoft Visual Studio is available then that can be used instead.

5.2 Install the sample

1. In the integration toolkit select **File**→**Switch Workspace**→**Other**.

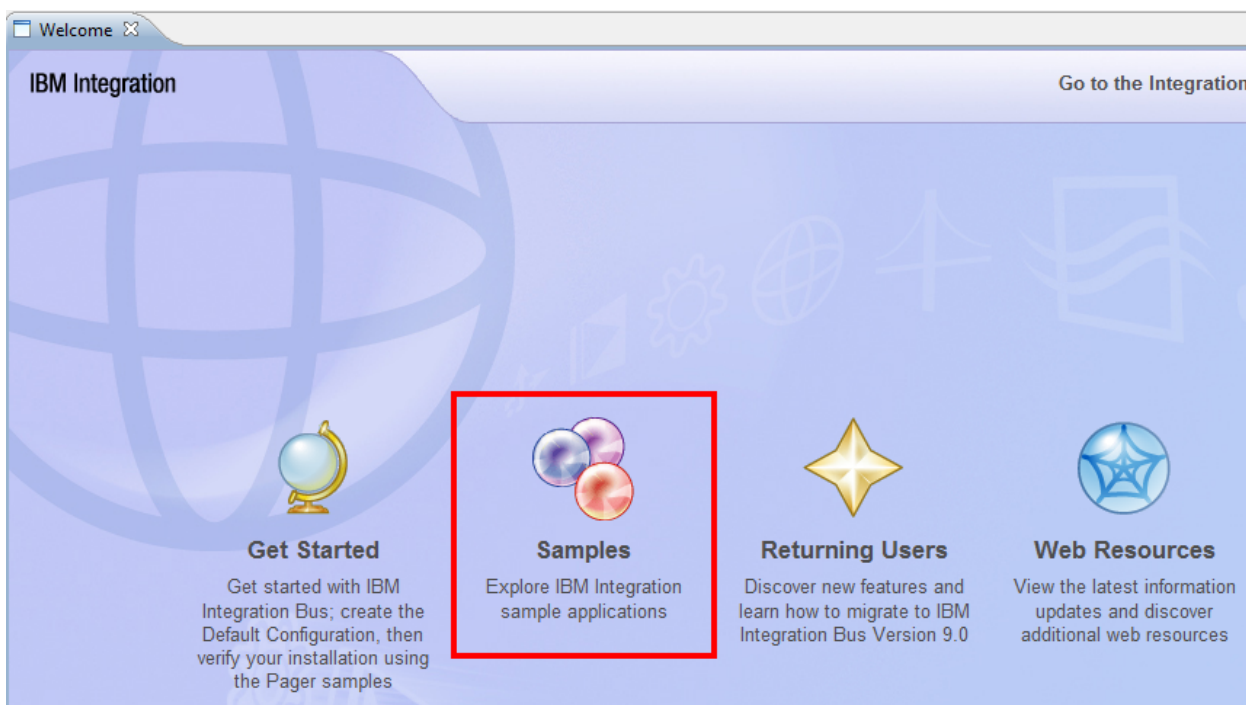


- __2. Select the **C:\student\dotNet\workspace** workspace (the **Browse** button can be used).
- __3. Press the **OK** button to continue.



A Welcome screen is displayed.

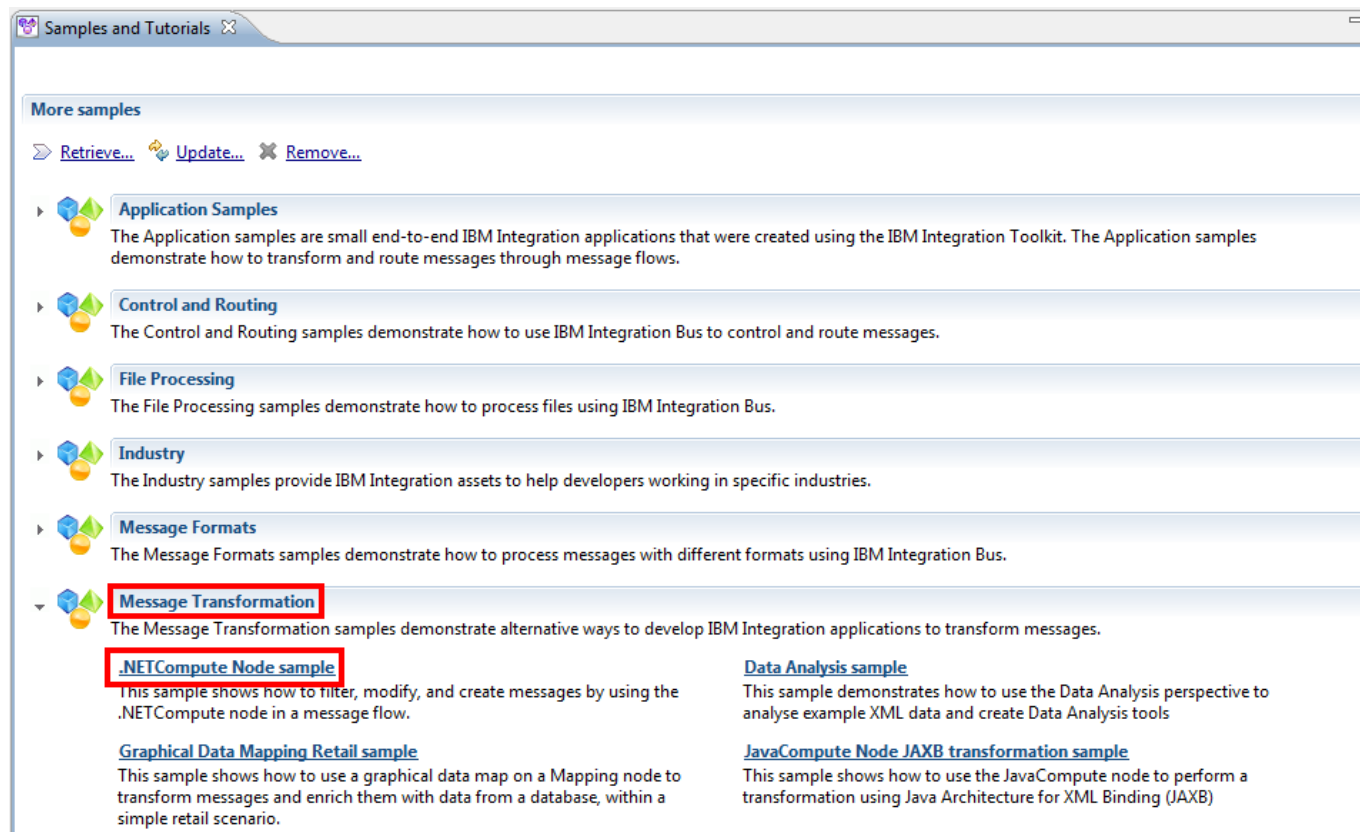
- __4. Click the **Samples** hot spot. (If the Welcome screen is not displayed, select **Help→Welcome**.)



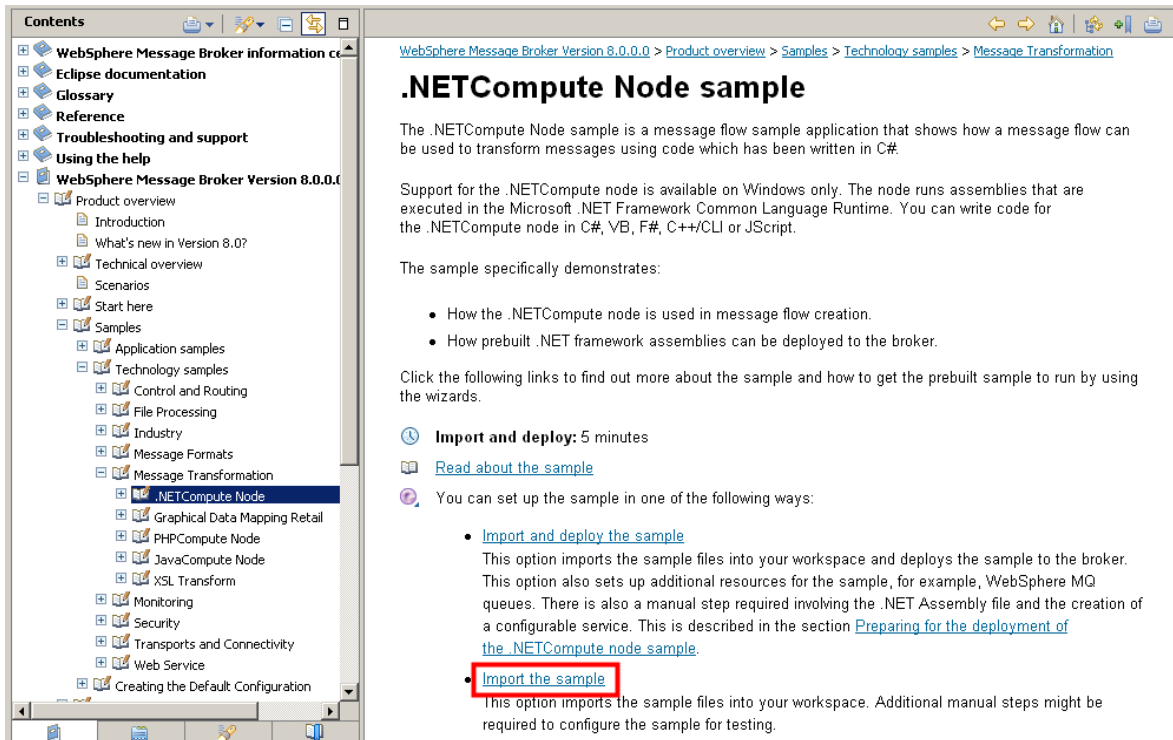
- __5. Close the **Welcome** tab.



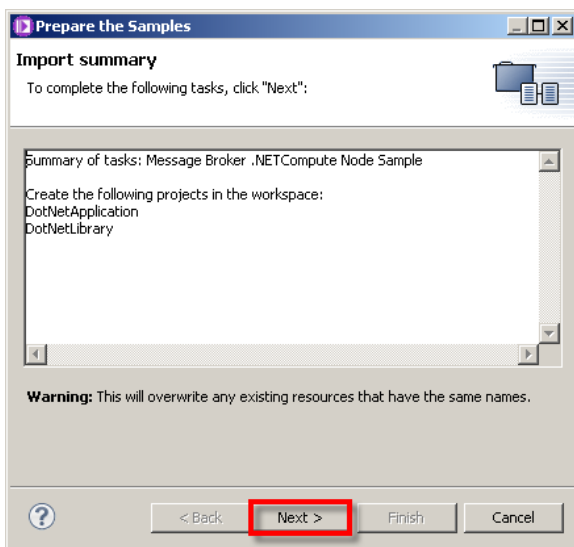
- __6. Scroll down so that the **Message Transformation** category is visible.
- __7. Expand the **Message Transformation** category.
- __8. Click the **.Net Compute Node sample**.



- __9. The sample preparation wizard should launch. Click on **Import the sample** (not Import and Deploy the sample).

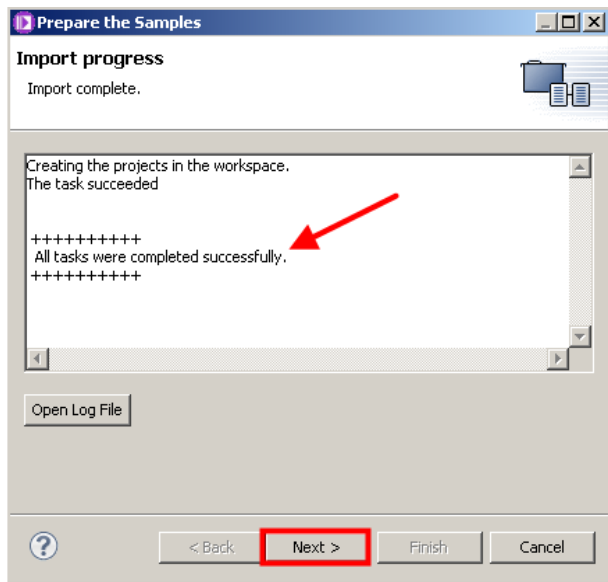


- __10. Press the **Next** button.

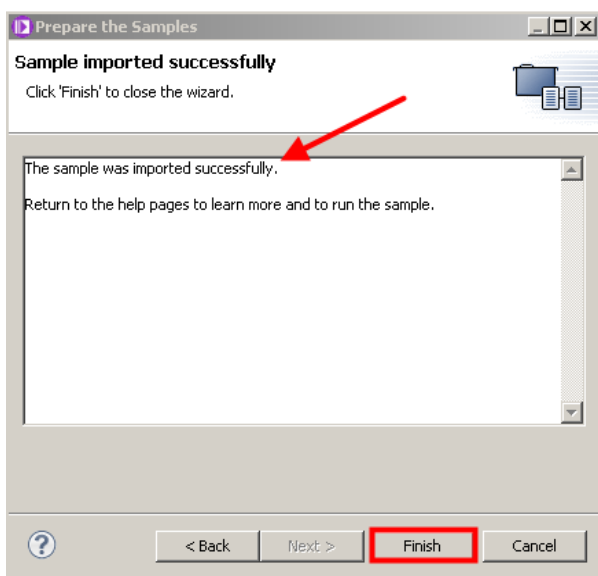


__11. Make sure that all the tasks were completed successfully.

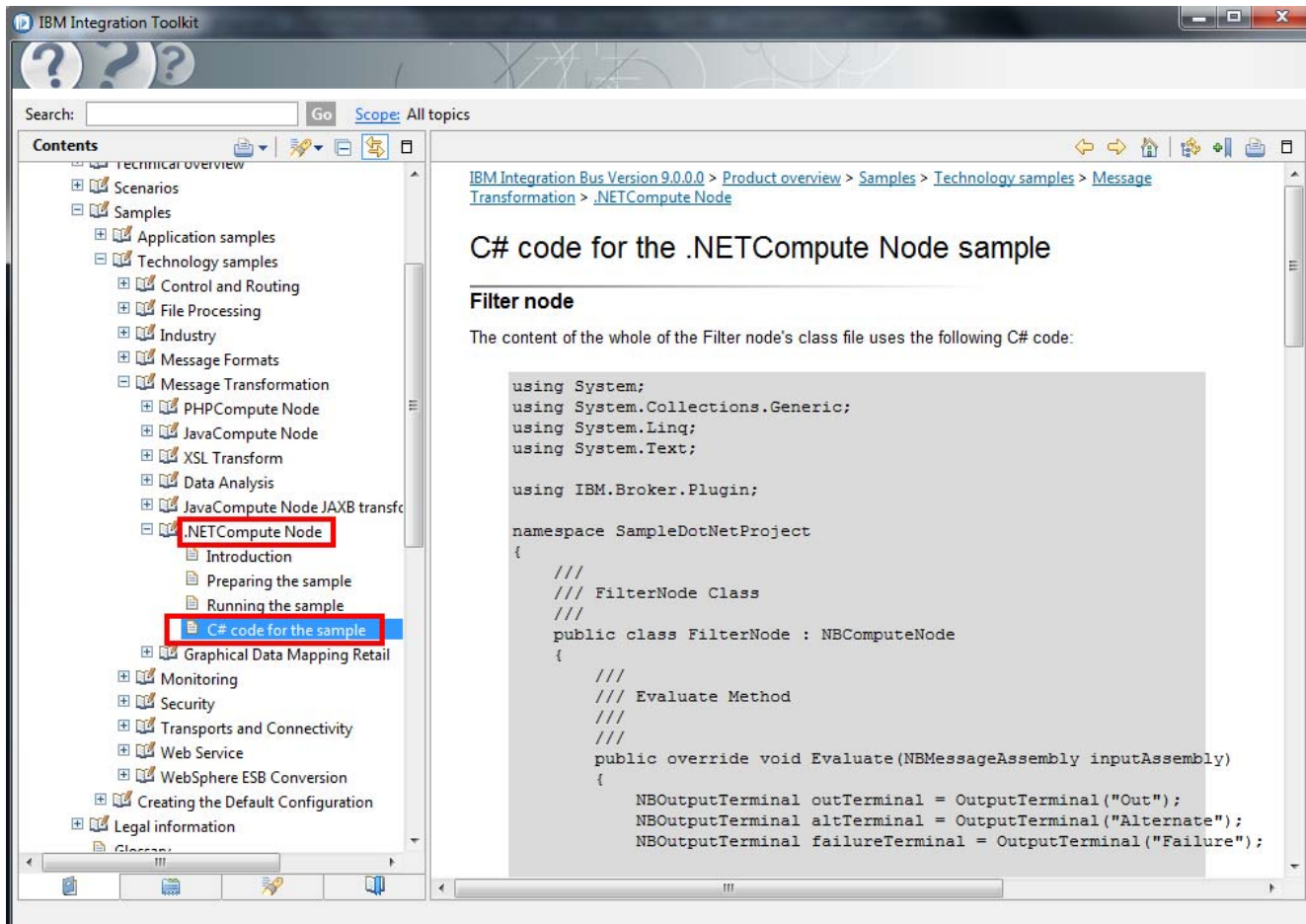
__12. Press the **Next** button.



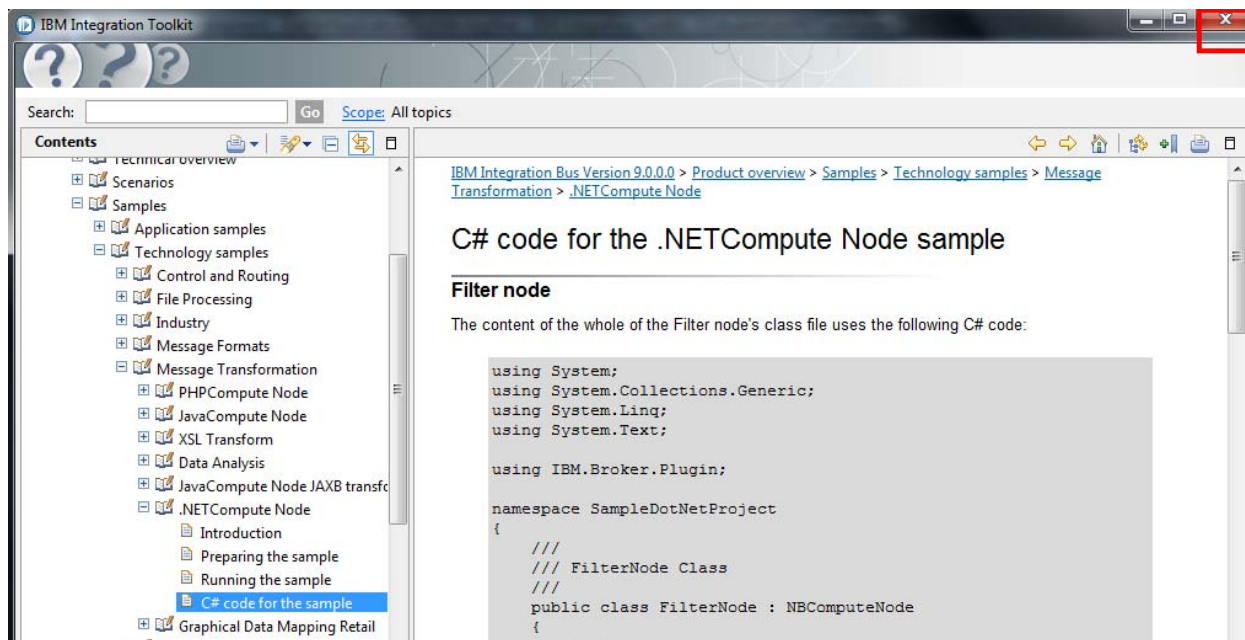
__13. Press the **Finish** button to dismiss the wizard.



- ___14. Control should return to the Information Center. Expand the **.NET Compute Node** entry.
- ___15. Select the **C# code for the sample** item.
- ___16. Examine the C# code for the nodes in the sample. There is further information available about the sample by clicking on the other items.



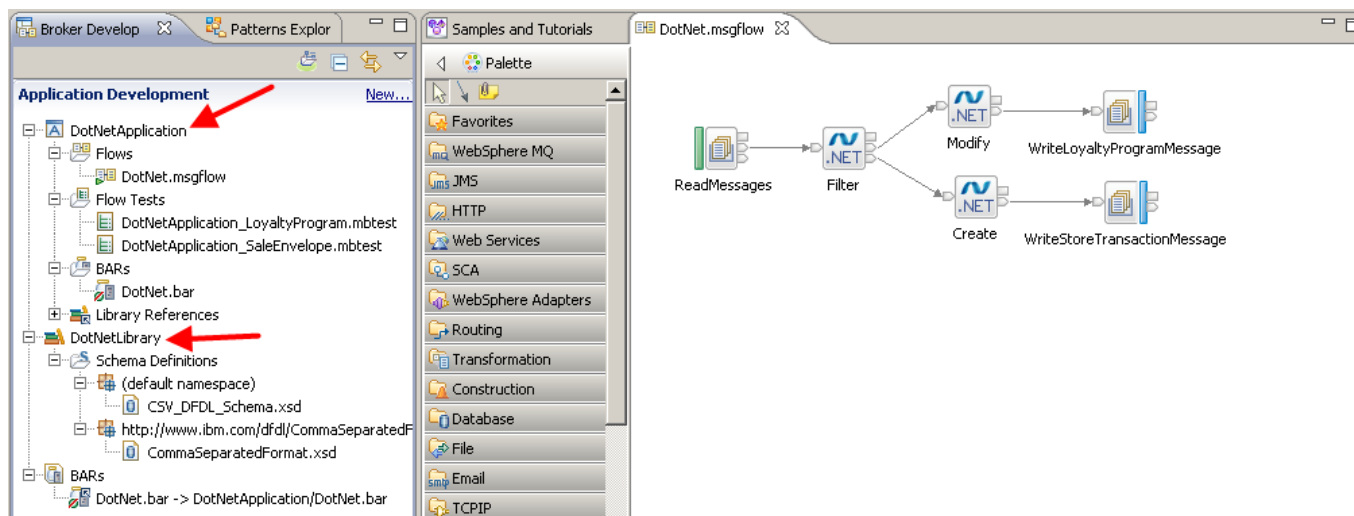
__17. After examining the sample documentation close the Information Center.



__18. Return to the integration toolkit.

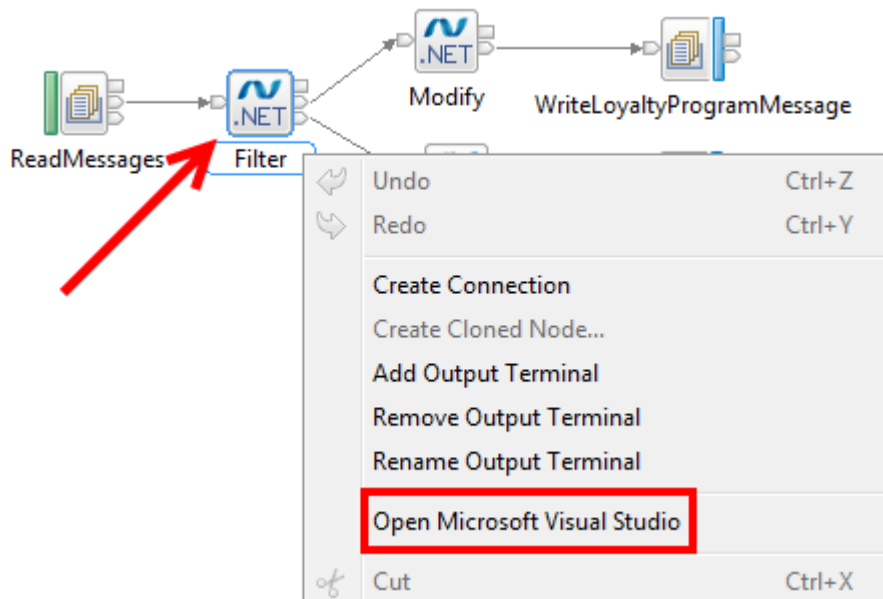
The sample has created a message flow in an application named DotNetApplication. The flow uses a library with a message model for a comma separated message format (CommaSeparatedFormat.xsd). Two message flow test cases and a broker archive are provided to test and deploy the message flow.

The message flow uses three .NET compute nodes. The first node (**Filter**) checks the input message format and routes the message to the correct path through the flow. The second node (**Modify**) adds some XML elements to the message and sends it to the loyalty application for processing. The third node (**Create**) transforms the input message into a new output message, which uses a comma separated variables (CSV) format. In both cases the output messages will be sent to the **DOTNET.OUT** queue. The loyalty path uses an MQ output node with the queue name specified in the node properties. The **Create** node creates a destination list with an entry for the **DOTNET.OUT** queue.

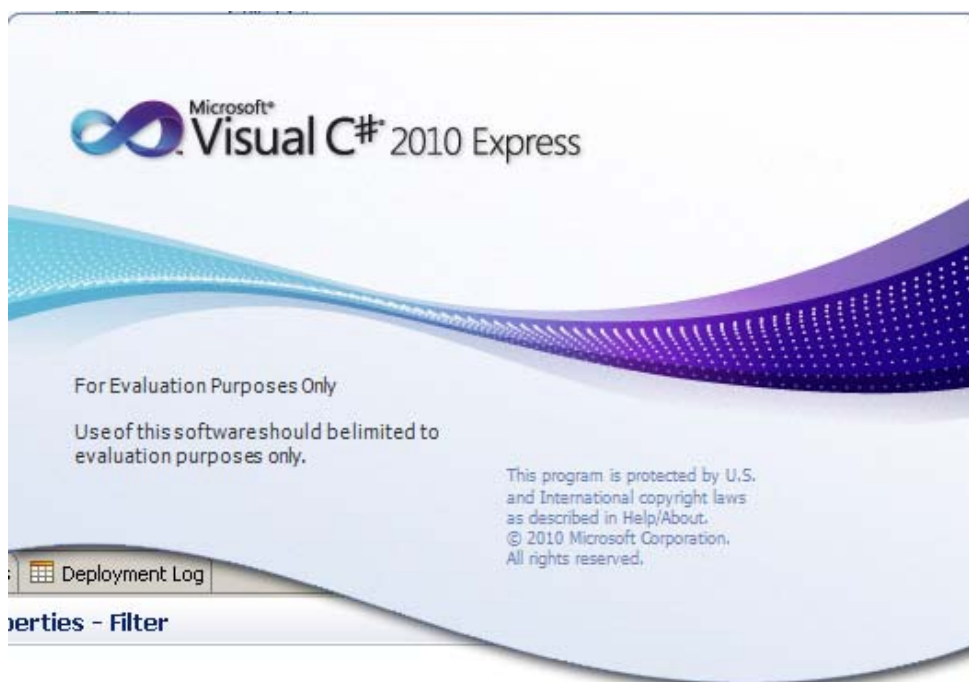


5.3 Create the .NET classes

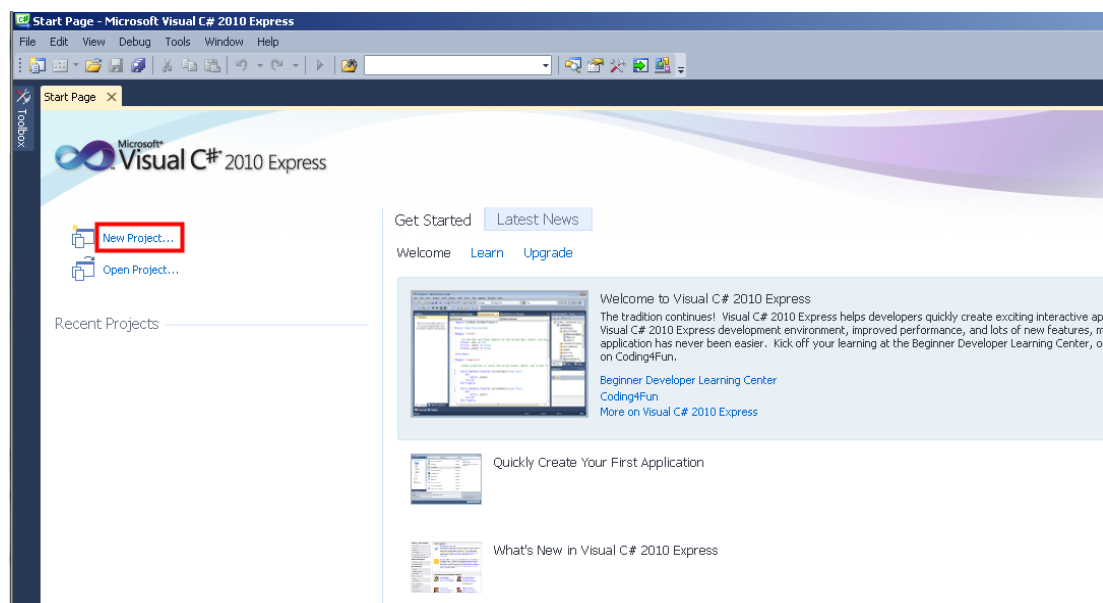
- __1. Select the .NET compute node labeled **Filter**.
- __2. Press the right mouse button.
- __3. Select **Open Microsoft Visual Studio** from the menu.



Visual Studio Express should start.



__4. Click on New Project...

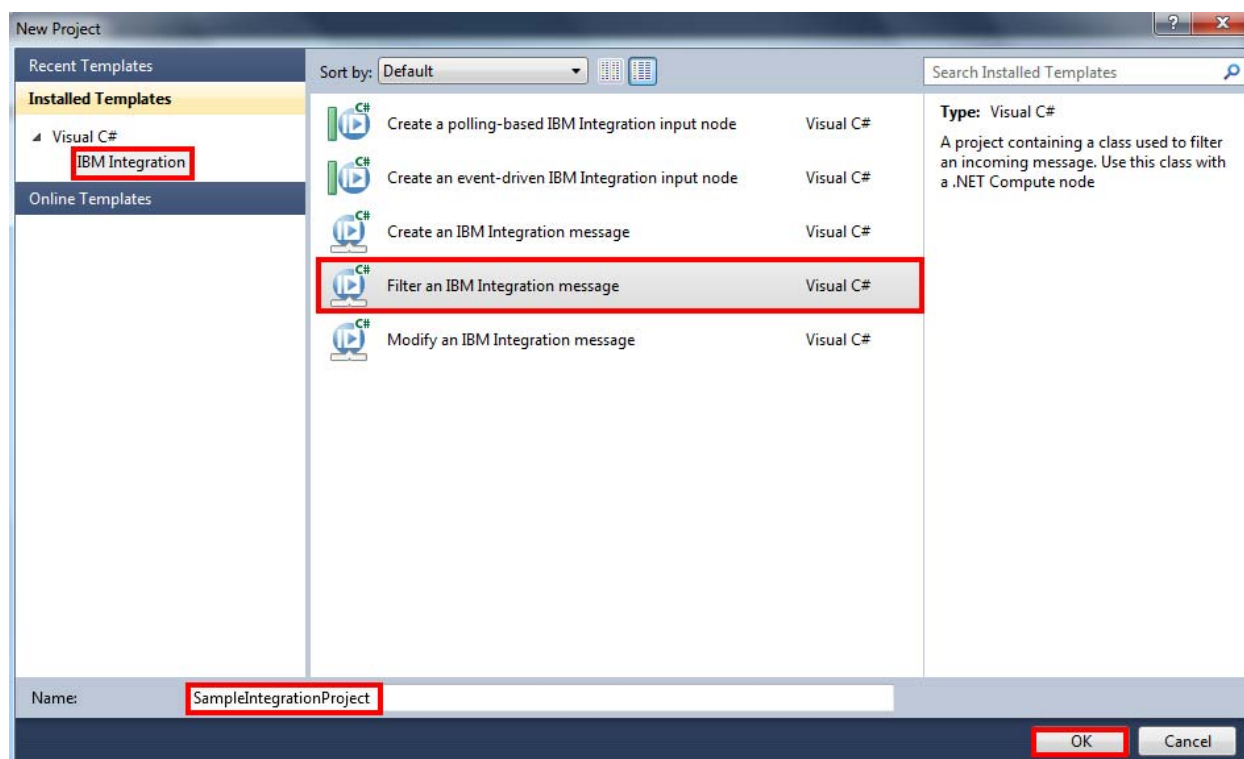


__5. Select **IBM Integration**.

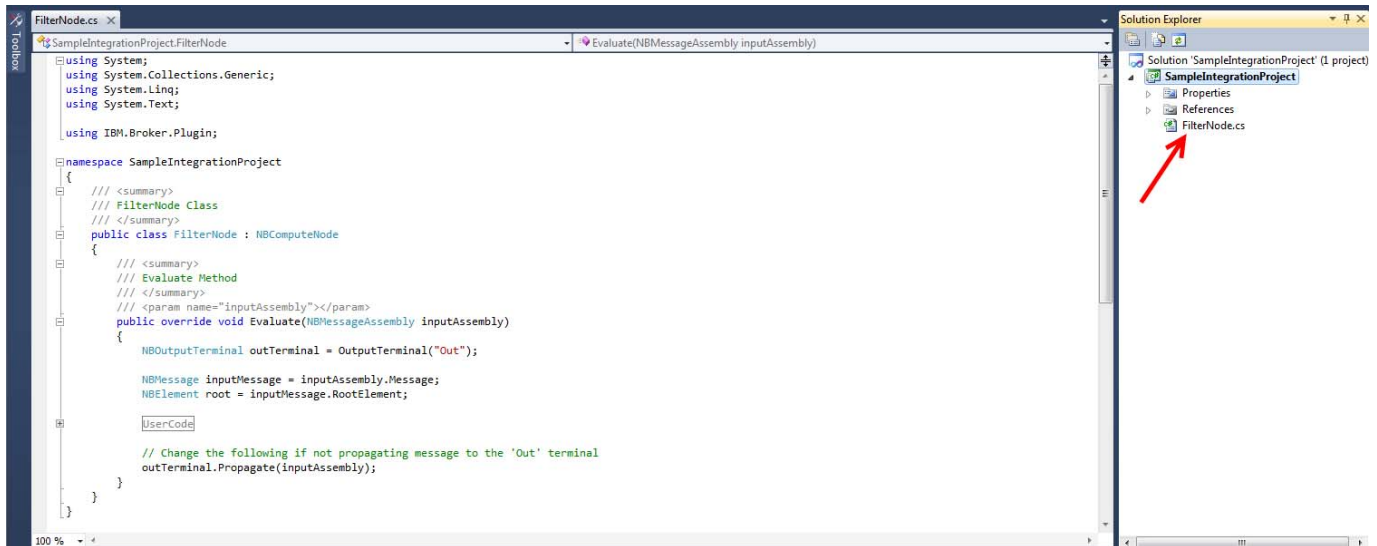
__6. Select the **Filter an IBM Integration message** template

__7. Enter **SampleIntegrationProject** as the **Name**.

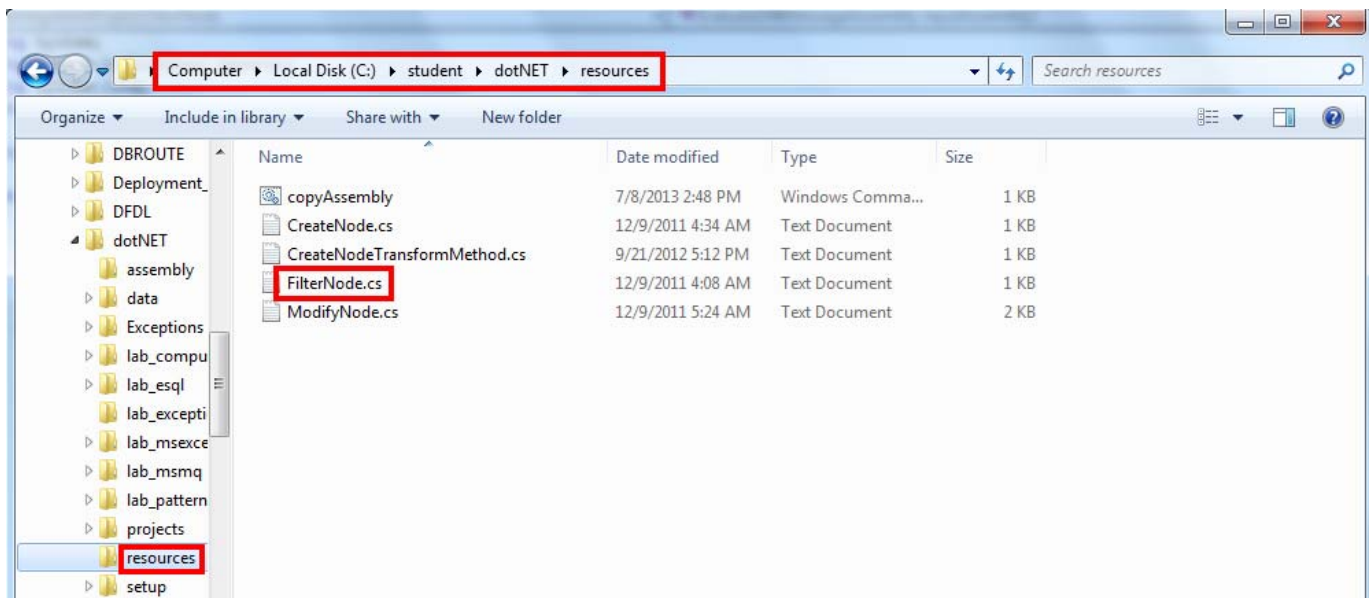
__8. Press the **OK** button.



The project should now be created. The C# editor should open with a skeleton program. The program (**FilterNode.cs**) should be visible in the **Solution Explorer** in the upper right.



9. Replace the contents of the **Evaluate** method with the code available in a text file called **FilterNode.cs** in the **C:\student\dotNET\resources** directory.



___10. Copy the code from **FilterNode.cs.txt** text file replacing the evaluate method in the template.

```
public class FilterNode : NBComputeNode
{
    /// <summary>
    /// Evaluate Method
    /// </summary>
    /// <param name="inputAssembly"></param>
    public override void Evaluate(NBMessageAssembly inputAssembly)
    {
        NBOutputTerminal outTerminal = OutputTerminal("Out");

        NBMessage inputMessage = inputAssembly.Message;
        NBElement root = inputMessage.RootElement;

        UserCode

        // Change the following if not propagating message to the 'Out' terminal
        outTerminal.Propagate(inputAssembly);
    }
}
```



11. Save the updated program (**Ctrl+S**).

```

FilterNode.cs x
SampleBrokerProject.FilterNode

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using IBM.Broker.Plugin;

namespace SampleBrokerProject
{
    /// <summary>
    /// FilterNode Class
    /// </summary>
    public class FilterNode : NBComputeNode
    {
        /// <summary>
        /// Evaluate Method
        /// </summary>
        /// <param name="inputAssembly"></param>
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("Out");
            NBOutputTerminal altTerminal = OutputTerminal("Alternate");
            NBOutputTerminal failureTerminal = OutputTerminal("Failure");

            NBMessage inputMessage = inputAssembly.Message;
            NBElement root = inputMessage.RootElement;

            #region UserCode
            // Add user code in this region to filter the message
            // The following expression deliberately uses LastChild in
            // preference to FirstChild in case an XML Declaration is present!
            switch (root[NBParsers.XMLNSC.ParserName].LastChild.Name)
            {
                case "LoyaltyProgram":
                    outTerminal.Propagate(inputAssembly);
                    break;
                case "SaleEnvelope":
                    altTerminal.Propagate(inputAssembly);
                    break;
                default:
                    failureTerminal.Propagate(inputAssembly);
                    break;
            }
            #endregion UserCode
        }
    }
}

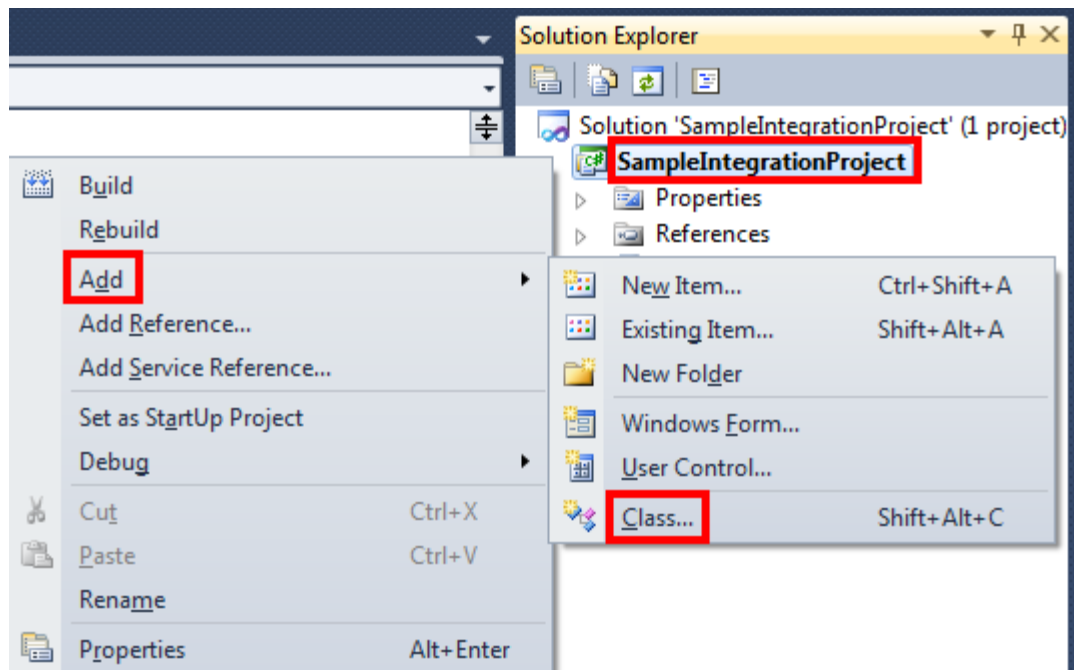
```

Use the **Solution Explorer** to add another **Class** to the **SampleIntegrationProject** project.

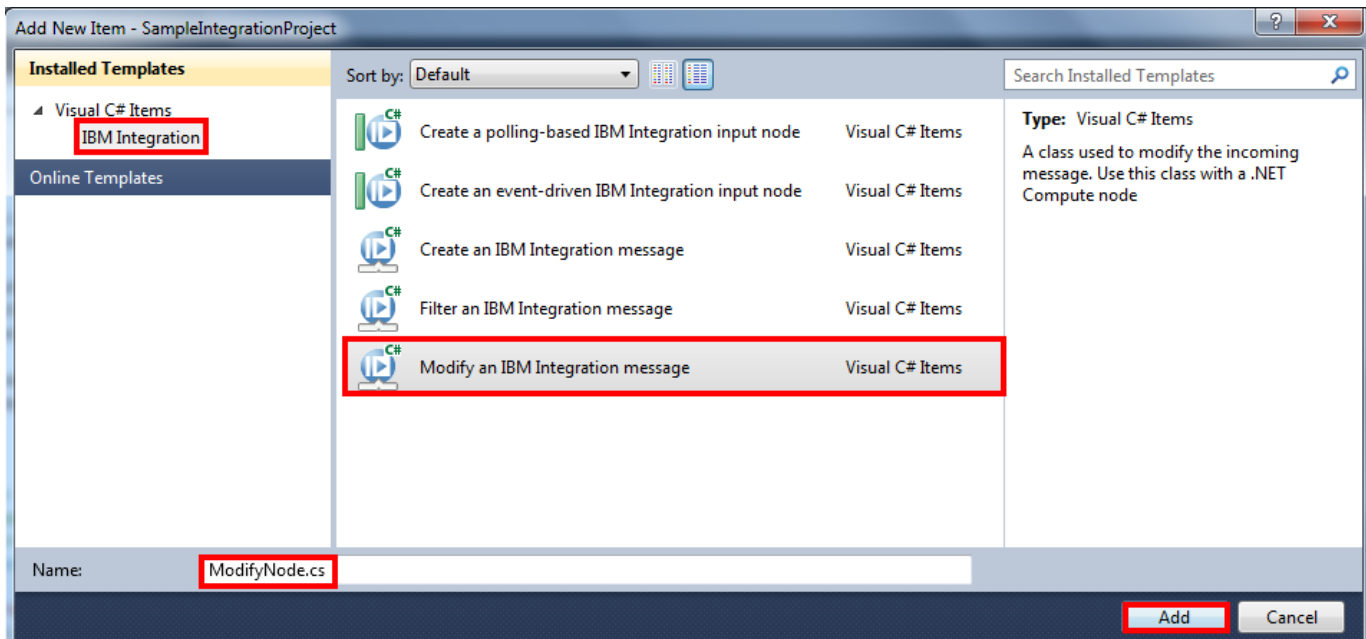
__12. In the **Solution Explorer** select the **SampleIntegrationProject** project.

__13. Press the right mouse button.

__14. Select **Add->Class** from the menu.

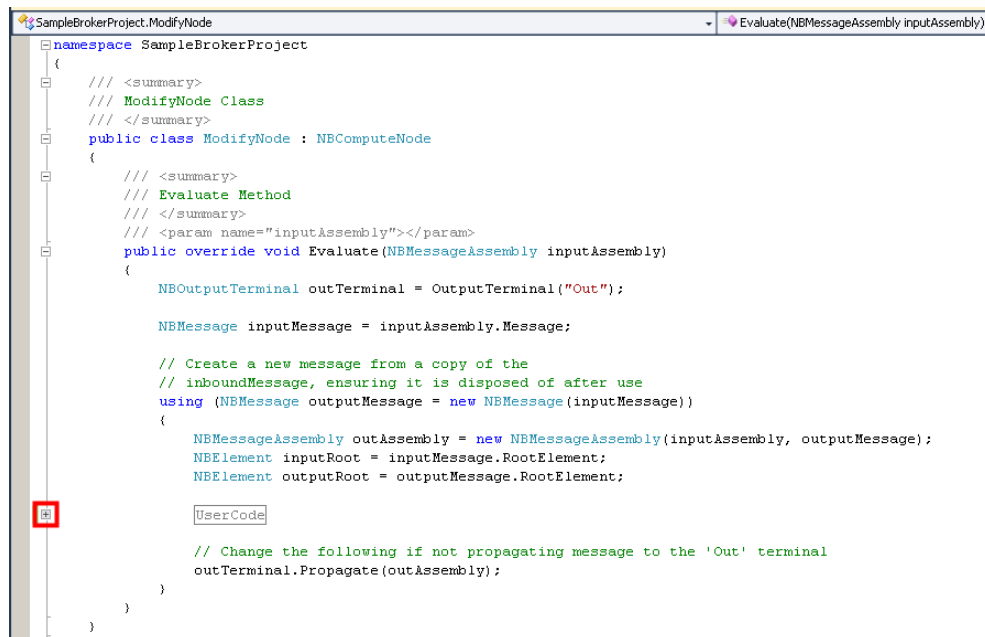


- __15. Select **IBM Integration**.
- __16. Select the **Modify an IBM Integration message** template.
- __17. Change the **Name** to **ModifyNode.cs**.
- __18. Press the **Add** button.



A skeleton module is created.

__19. Expand the **UserCode** area by clicking the small plus sign.



```

namespace SampleBrokerProject
{
    /// <summary>
    /// ModifyNode Class
    /// </summary>
    public class ModifyNode : NBComputeNode
    {
        /// <summary>
        /// Evaluate Method
        /// </summary>
        /// <param name="inputAssembly"></param>
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("Out");

            NBMessage inputMessage = inputAssembly.Message;

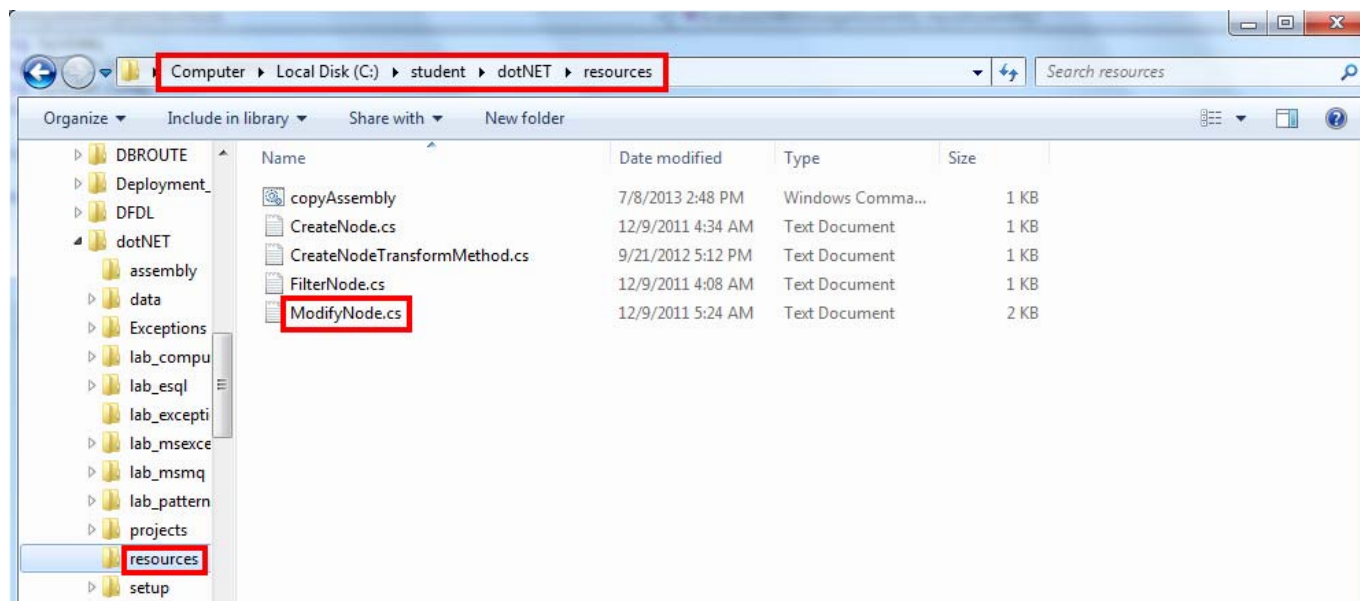
            // Create a new message from a copy of the
            // inboundMessage, ensuring it is disposed of after use
            using (NBMessage outputMessage = new NBMessage(inputMessage))
            {
                NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly, outputMessage);
                NBElement inputRoot = inputMessage.RootElement;
                NBElement outputRoot = outputMessage.RootElement;

                UserCode

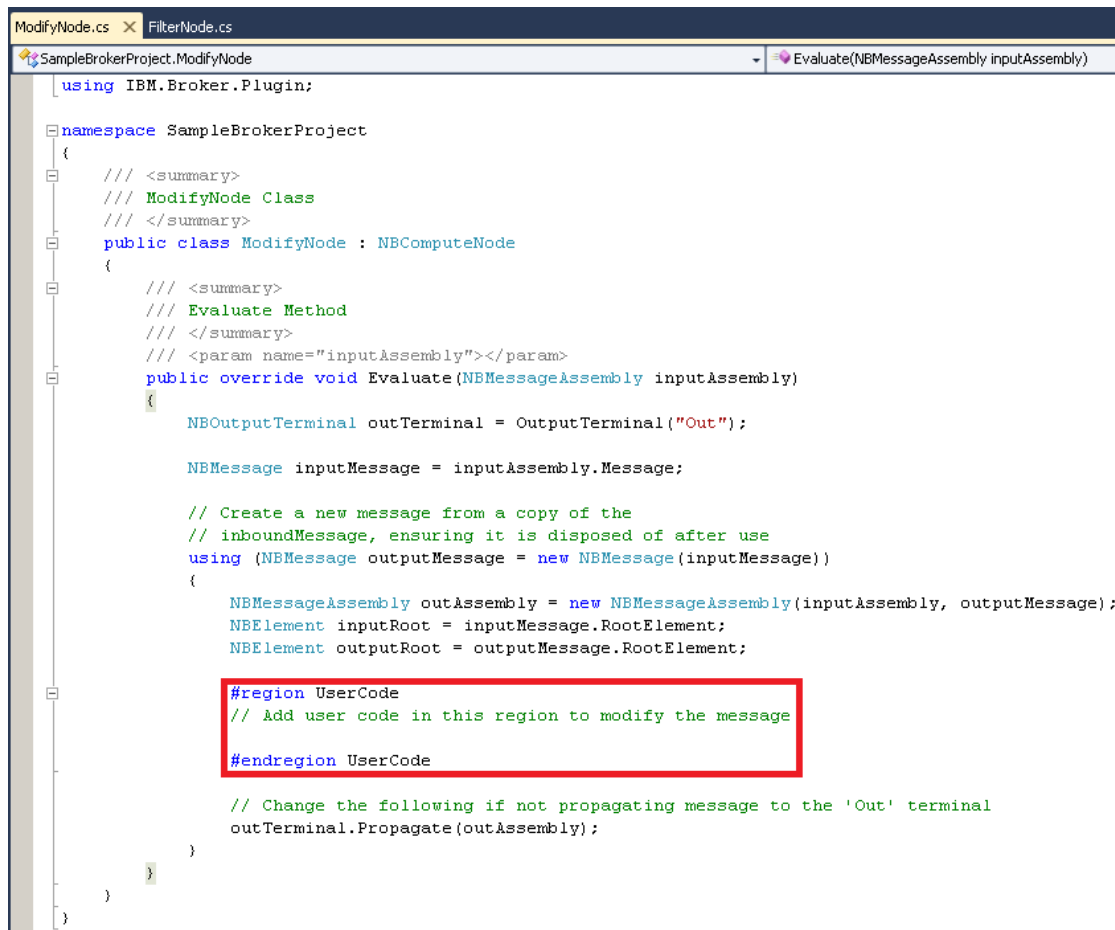
                // Change the following if not propagating message to the 'Out' terminal
                outTerminal.Propagate(outAssembly);
            }
        }
    }
}

```

__20. Replace the contents of the **UserCode** region with the code available in a file called **ModifyNode.cs.txt** in the **C:\student\dotNET\resources** directory.



Replace the highlighted code:



```

ModifyNode.cs x FilterNode.cs
SampleBrokerProject.ModifyNode Evaluate(NBMessageAssembly inputAssembly)
using IBM.Broker.Plugin;

namespace SampleBrokerProject
{
    /// <summary>
    /// ModifyNode Class
    /// </summary>
    public class ModifyNode : NBComputeNode
    {
        /// <summary>
        /// Evaluate Method
        /// </summary>
        /// <param name="inputAssembly"></param>
        public override void Evaluate(NBMessageAssembly inputAssembly)
        {
            NBOutputTerminal outTerminal = OutputTerminal("Out");

            NBMessage inputMessage = inputAssembly.Message;

            // Create a new message from a copy of the
            // inboundMessage, ensuring it is disposed of after use
            using (NBMessage outputMessage = new NBMessage(inputMessage))
            {
                NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly, outputMessage);
                NBElement inputRoot = inputMessage.RootElement;
                NBElement outputRoot = outputMessage.RootElement;

                #region UserCode
                // Add user code in this region to modify the message
                #endregion UserCode

                // Change the following if not propagating message to the 'Out' terminal
                outTerminal.Propagate(outAssembly);
            }
        }
    }
}

```



21.

Save the updated program (**Ctrl+S**).

```

ModifyNode.cs  X  FilterNode.cs
SampleBrokerProject.ModifyNode  Evaluate(NBMessage)

NBMessage inputMessage = inputAssembly.Message;

// Create a new message from a copy of the
// inboundMessage, ensuring it is disposed of after use
using (NBMessage outputMessage = new NBMessage(inputMessage))
{
    NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly, outputMessage);
    NBElement inputRoot = inputMessage.RootElement;
    NBElement outputRoot = outputMessage.RootElement;

    #region UserCode
    NBElement xmlRoot = outputRoot[NBParasers.XMLNSC.ParserName];
    NBElement xmlDecl = xmlRoot[NBParasers.XMLNSC.XmlDeclaration, "XmlDeclaration"];
    if (xmlDecl == null)
    {
        // Create an XML Declaration if required
        NBParasers.XMLNSC.CreateXmlDeclaration(xmlRoot, "1.0", "UTF-8", "yes");
    }
    string notarget = "";
    string ns = "http://www.example.org/store";
    NBElement storeDetails = xmlRoot[notarget, "LoyaltyProgram"][ns, "StoreDetails"];
    string storeName = "";
    string storeStreet = "";
    string storeTown = "Happyville";
    switch ((string)storeDetails[ns, "StoreID"])
    {
        case "001":
            storeName = "Broker Brothers Central";
            storeStreet = "Exuberant Avenue";
            break;
        case "002":
            storeName = "Broker Brothers Mall";
            storeStreet = "Enthusiastic Crescent";
            break;
        case "003":
            storeName = "Broker Brothers District";
            storeStreet = "Peaceful Road";
            break;
    }
    storeDetails.CreateLastChild(ns, "StoreName", storeName);
    storeDetails.CreateLastChild(ns, "StoreStreet", storeStreet);
    storeDetails.CreateLastChild(ns, "StoreTown", storeTown);
    #endregion UserCode

    // Change the following if not propagating message to the 'Out' terminal
    outTerminal.Propagate(outAssembly);
}
}
}

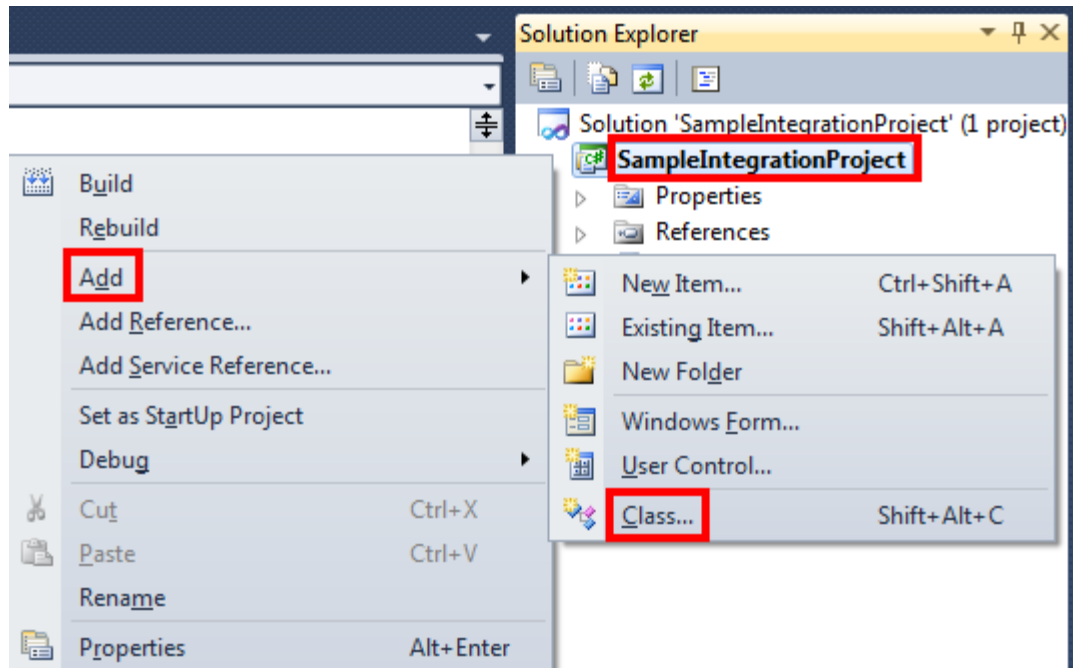
```


Use the **Solution Explorer** to add another **Class** to the **SampleIntegrationProject** project.

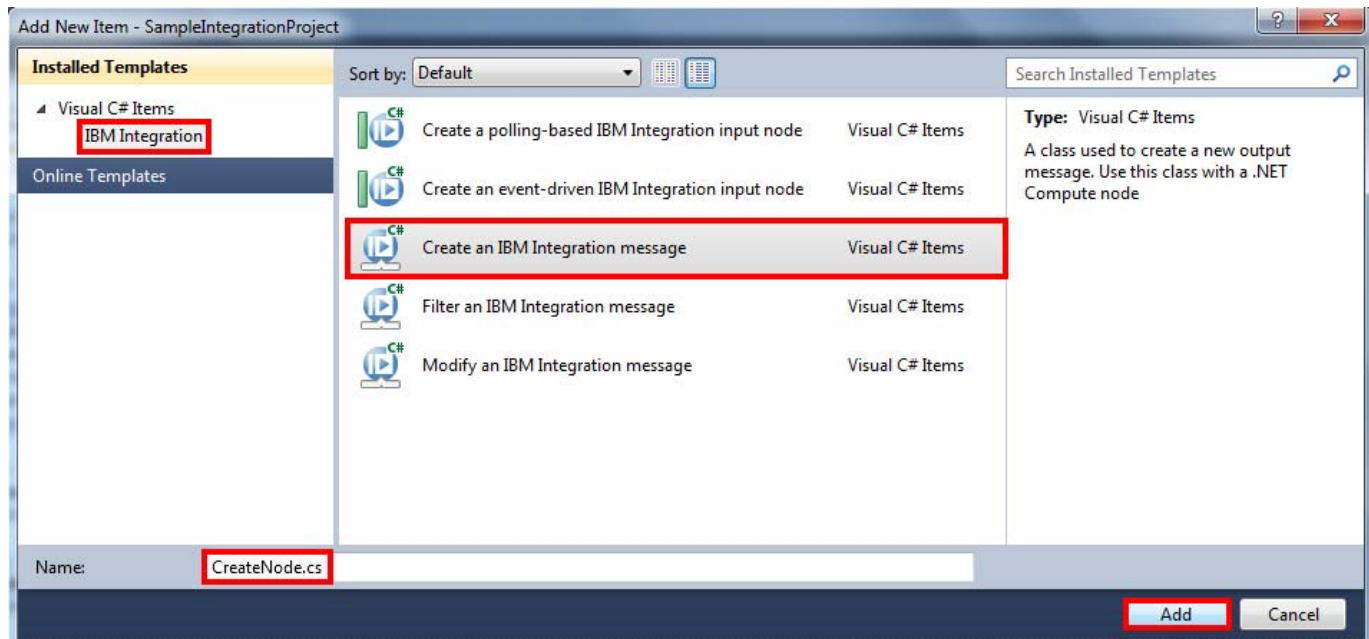
__22. In the **Solution Explorer** select the **SampleIntegrationProject** project.

__23. Press the right mouse button.

__24. Select **Add->Class** from the menu.

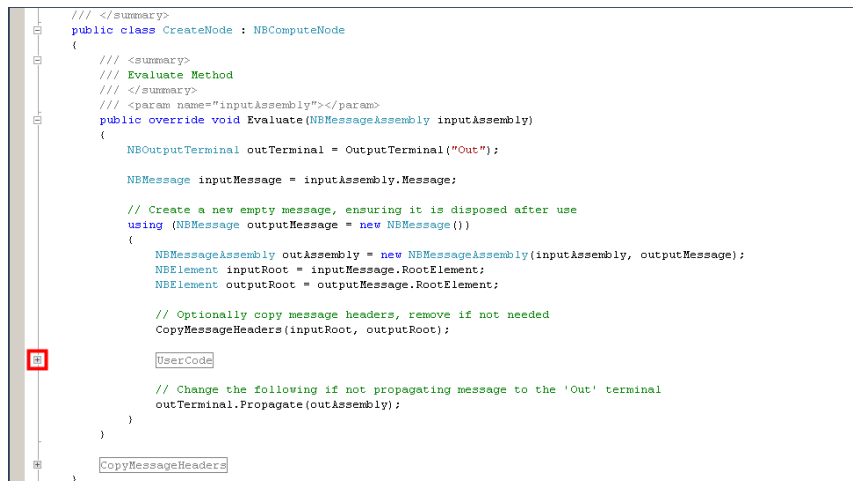


- __25. Select **IBM Integration**.
- __26. Select the **Create an IBM Integration message** template.
- __27. Change the **Name** to **CreateNode.cs**.
- __28. Press the **Add** button.

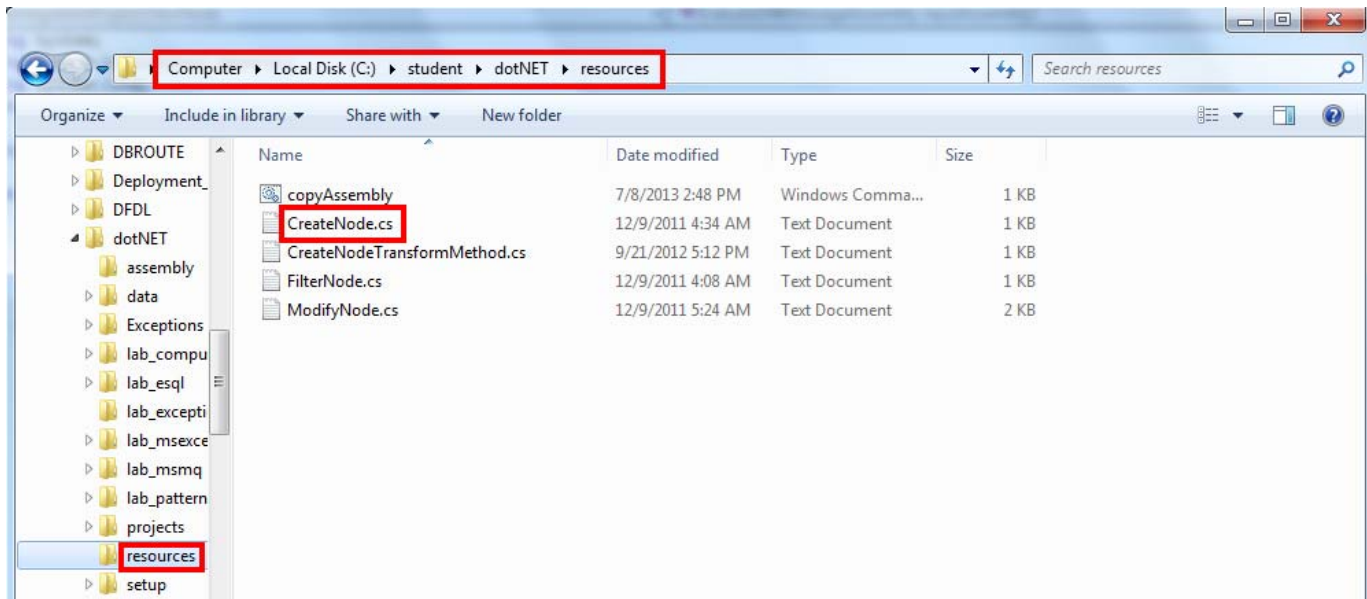


A skeleton module is created.

- __29. Expand the **UserCode** area by clicking the small plus sign.



- ___30. Replace the contents of the **UserCode** region with the code below. This code is available in a file called **CreateNode.cs** in the **C:\student\dotNETresources** directory.



Replace the highlighted code:

```

CreateNode.cs | ModifyNode.cs | FilterNode.cs
SampleBrokerProject.CreateNode
CopyMessageHeaders(NBElement inputRoot, NBElement outputRoot)

/// CreateNode Class
/// </summary>
public class CreateNode : NBComputeNode
{
    /// <summary>
    /// Evaluate Method
    /// </summary>
    /// <param name="inputAssembly"></param>
    public override void Evaluate(NBMessageAssembly inputAssembly)
    {
        NBOutputTerminal outTerminal = OutputTerminal("Out");

        NBMessage inputMessage = inputAssembly.Message;

        // Create a new empty message, ensuring it is disposed after use
        using (NBMessage outputMessage = new NBMessage())
        {
            NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly, outputMessage);
            NBElement inputRoot = inputMessage.RootElement;
            NBElement outputRoot = outputMessage.RootElement;

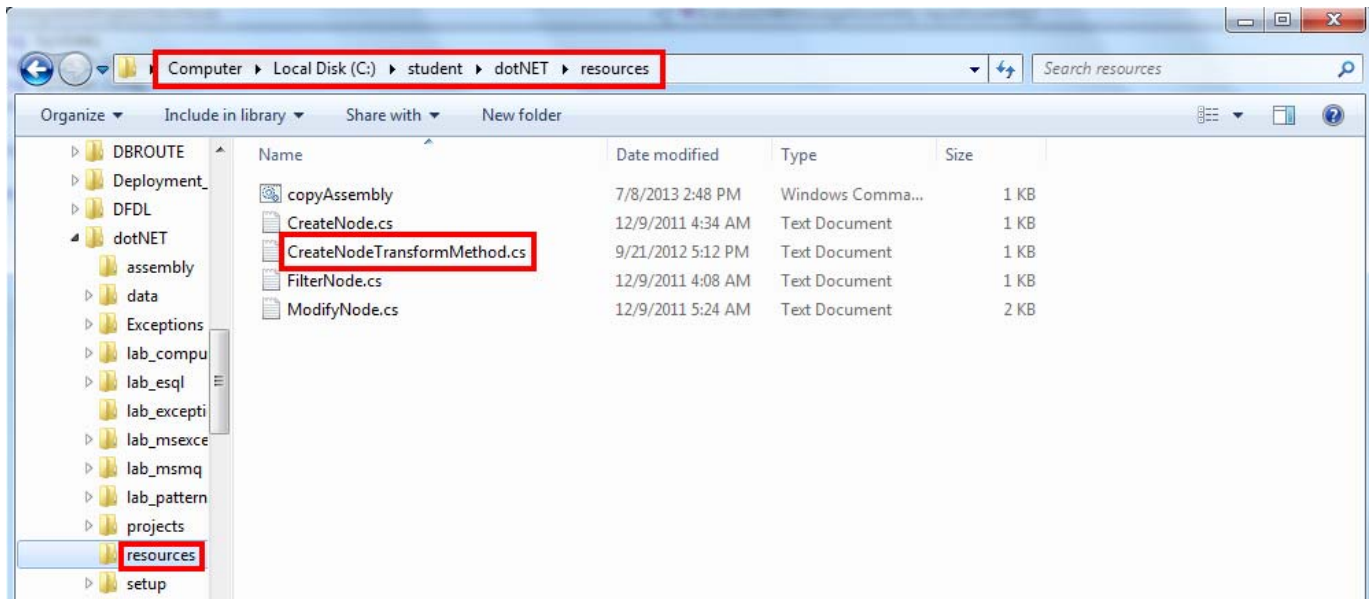
            // Optionally copy message headers, remove if not needed
            CopyMessageHeaders(inputRoot, outputRoot);

            #region UserCode
            // Add user code in this region to create a new output message
            #endregion UserCode

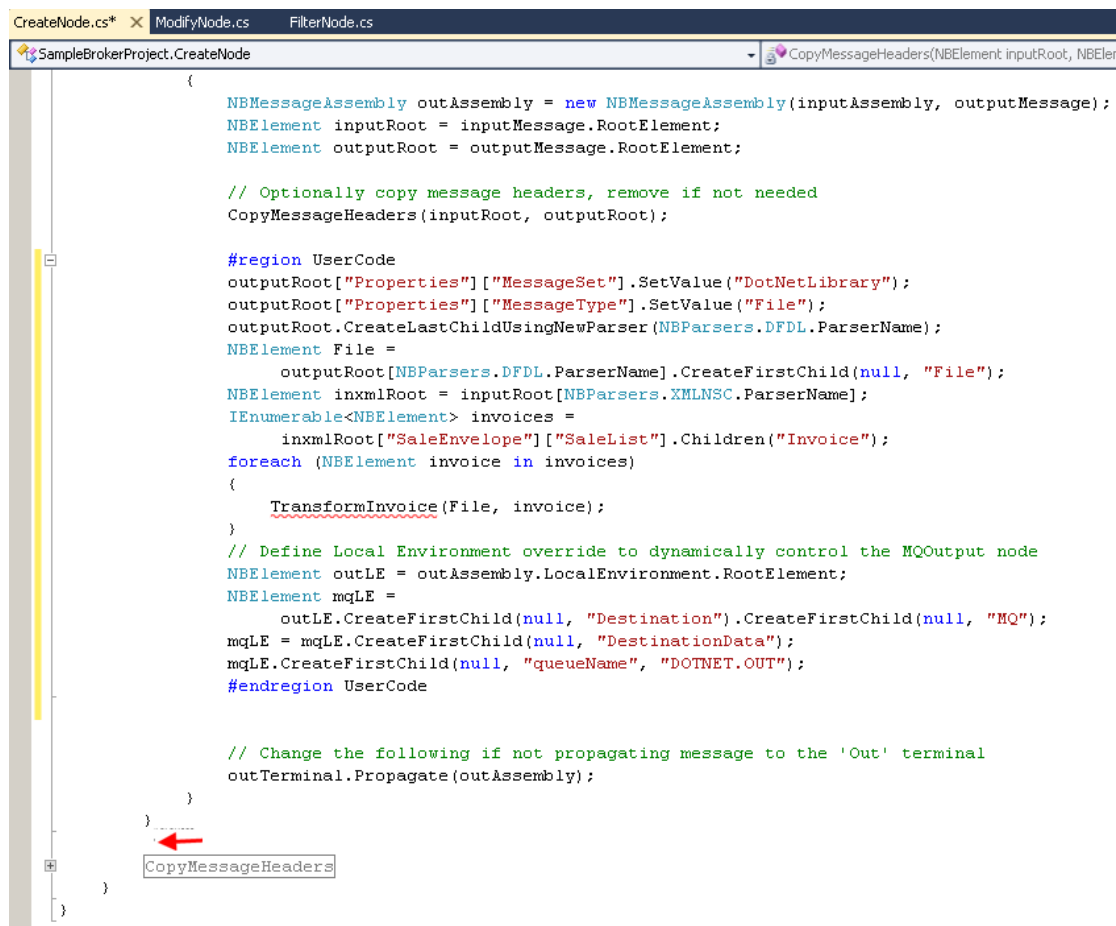
            // Change the following if not propagating message to the 'Out' terminal
            outTerminal.Propagate(outAssembly);
        }
    }
}
CopyMessageHeaders

```

- ___31. Insert the routine **CreateNodeTransformMethod.cs** above the **CopyMessageHeaders** region. This code is available in a file called **CreateNodeTransformMethod.cs.txt** in the **C:\student\dotNET\resources** directory.



Insert the code at the location pointed to by the arrow:



```

CreateNode.cs* x ModifyNode.cs FilterNode.cs
SampleBrokerProject.CreateNode CopyMessageHeaders(NBElement inputRoot, NBElement outputRoot)

{
    NBMessageAssembly outAssembly = new NBMessageAssembly(inputAssembly, outputMessage);
    NBElement inputRoot = inputMessage.RootElement;
    NBElement outputRoot = outputMessage.RootElement;

    // Optionally copy message headers, remove if not needed
    CopyMessageHeaders(inputRoot, outputRoot);

    #region UserCode
    outputRoot["Properties"]["MessageSet"].SetValue("DotNetLibrary");
    outputRoot["Properties"]["MessageType"].SetValue("File");
    outputRoot.CreateLastChildUsingNewParser(NBParsers.DFDL.ParserName);
    NBElement File =
        outputRoot[NBParsers.DFDL.ParserName].CreateFirstChild(null, "File");
    NBElement inxmlRoot = inputRoot[NBParsers.XMLNSC.ParserName];
    IEnumerable<NBElement> invoices =
        inxmlRoot["SaleEnvelope"]["SaleList"].Children("Invoice");
    foreach (NBElement invoice in invoices)
    {
        TransformInvoice(File, invoice);
    }
    // Define Local Environment override to dynamically control the MQOutput node
    NBElement outLE = outAssembly.LocalEnvironment.RootElement;
    NBElement mqLE =
        outLE.CreateFirstChild(null, "Destination").CreateFirstChild(null, "MQ");
    mqLE = mqLE.CreateFirstChild(null, "DestinationData");
    mqLE.CreateFirstChild(null, "queueName", "DOTNET.OUT");
    #endregion UserCode

    // Change the following if not propagating message to the 'Out' terminal
    outTerminal.Propagate(outAssembly);
}
CopyMessageHeaders

```



__32. Save the updated program (**Ctrl+S**).

```

CreateNode.cs x ModifyNode.cs FilterNode.cs
SampleBrokerProject.CreateNode CopyMessageHeaders(NBElement in

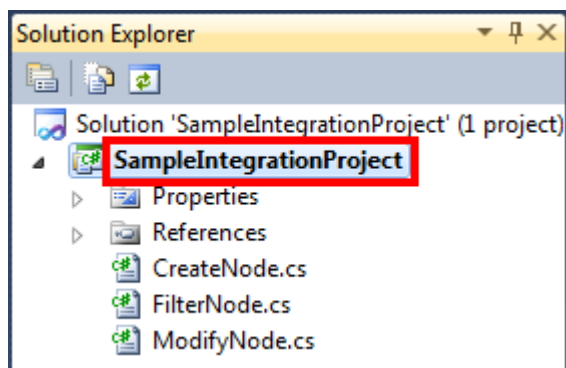
mqLE.CreateFirstChild(null, "queueName", "DOTNET.OUT");
#endregion UserCode

// Change the following if not propagating message to the 'Out' terminal
outTerminal.Propagate(outAssembly);
}
}
#region TransformInvoice
private static void TransformInvoice(NBElement outFileEl, NBElement inInvEl)
{
    // This method creates a structure based on
    // the Invoice Element in the input message
    IEnumerable<NBElement> items = inInvEl.Children("Item");
    foreach (NBElement item in items)
    {
        NBElement record = outFileEl.CreateLastChild(null, "Record");
        string notgt = "";
        record.CreateLastChild(notgt, "Code1", (string) item["Code", 0]);
        record.CreateLastChild(notgt, "Code2", (string) item["Code", 1]);
        record.CreateLastChild(notgt, "Code3", (string) item["Code", 2]);
        record.CreateLastChild(notgt, "Description", (string) item["Description"]);
        record.CreateLastChild(notgt, "Category", (string) item["Category"]);
        record.CreateLastChild(notgt, "Price", (decimal) item["Price"]);
        record.CreateLastChild(notgt, "Quantity", (Int32) item["Quantity"]);
    }
}
#endregion TransformInvoice
CopyMessageHeaders
}
}

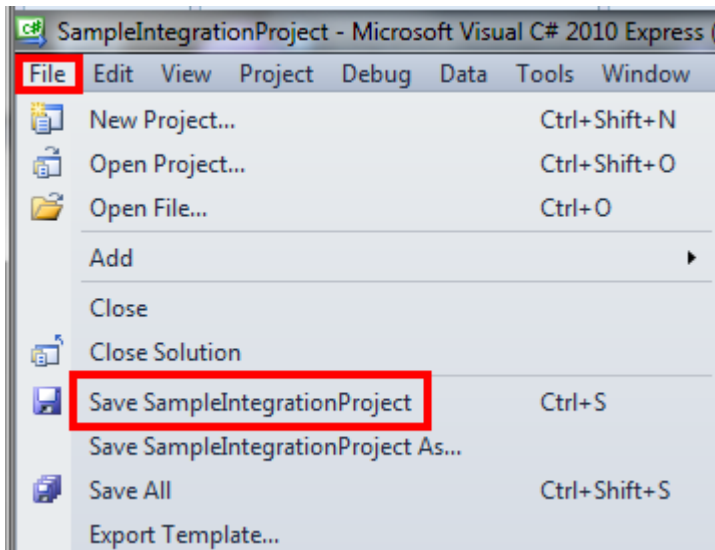
```

__33. Now In the **Solution Explorer**, select the **SampleIntegrationProject** project.

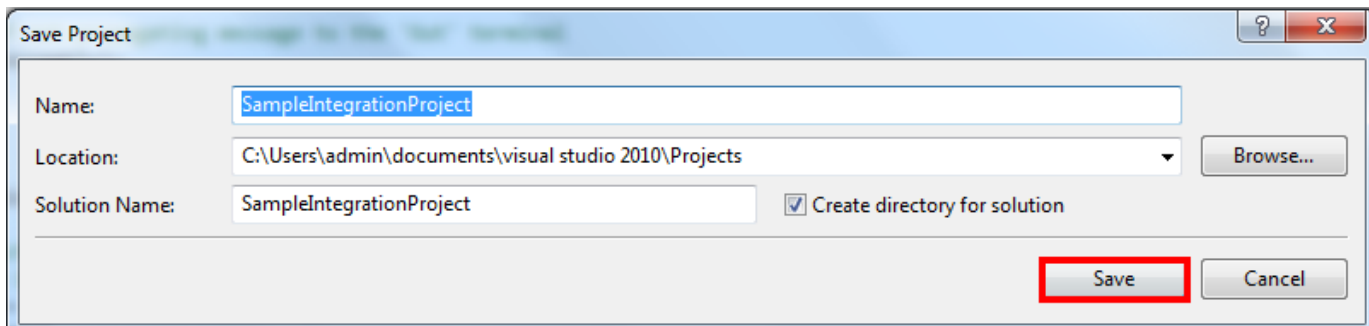
There should be three classes visible in the project.



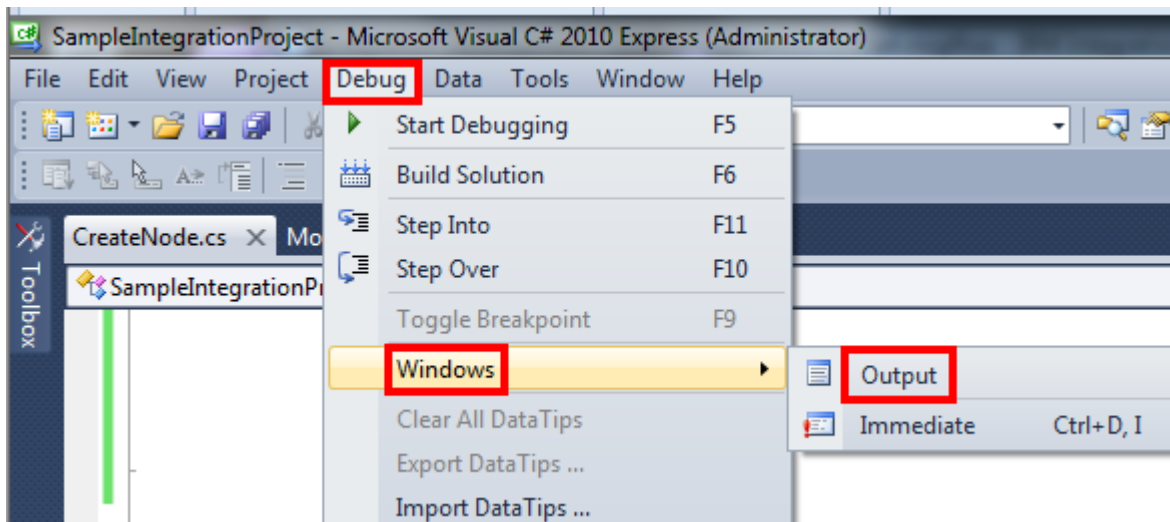
__34. In the project menus, click **File→Save SampleIntegrationProject**.



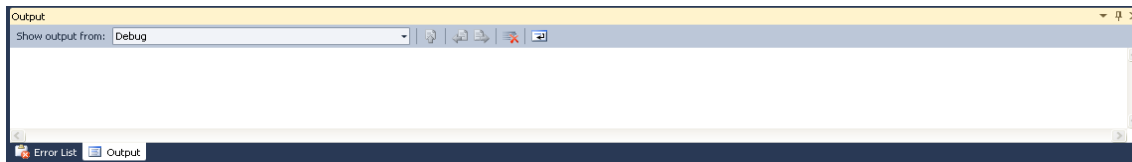
__35. In the Save Project dialog, take the defaults and click **Save**.



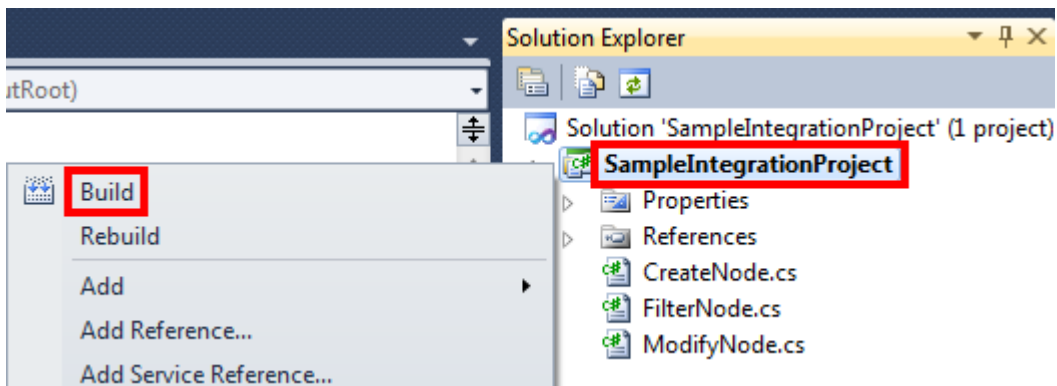
__36. In the project menus, select **Debug→Windows→Output**.



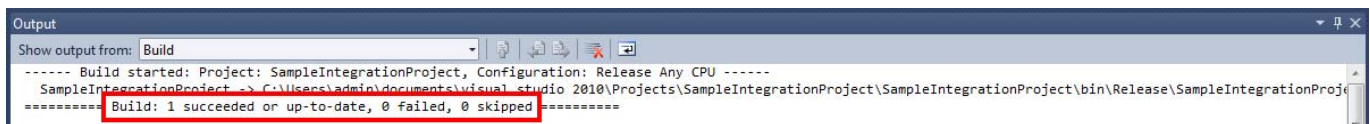
Output Window:



- ___37. In the **Solution Explorer** pane select the **SampleIntegrationProject** project.
- ___38. Press the right mouse button.
- ___39. Select **Build** from the menu.



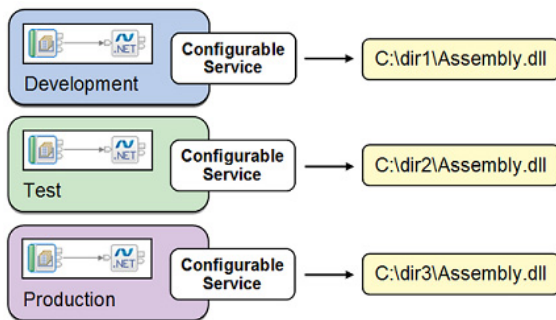
The results of the build operation should be visible in the **Output** window.



- ___40. Close Visual Studio.

5.4 Create the configurable service

Having built an assembly file from the C# code, it is possible to drag the assembly file from a Windows Explorer window directly onto a .NET compute node in the integration toolkit in order to associate the code with the node. This technique results in a hard-coded absolute file system location pointing to the assembly. It is useful when developing, testing and hot-swapping the .NET code that the node is executing. However, for production situations, a better approach is to define a configurable service that specifies the location of the assembly file. This method is more dynamic and better suited when moving a deployment between environments during development, test, and production. The diagram below shows that the same message flow can be used in multiple environments, with the configurable service defining the location of the assembly file, which might be at a different location for each environment.



Windows Explorer will be used to copy the assembly files to a known location.

__1. Navigate to the following directory.

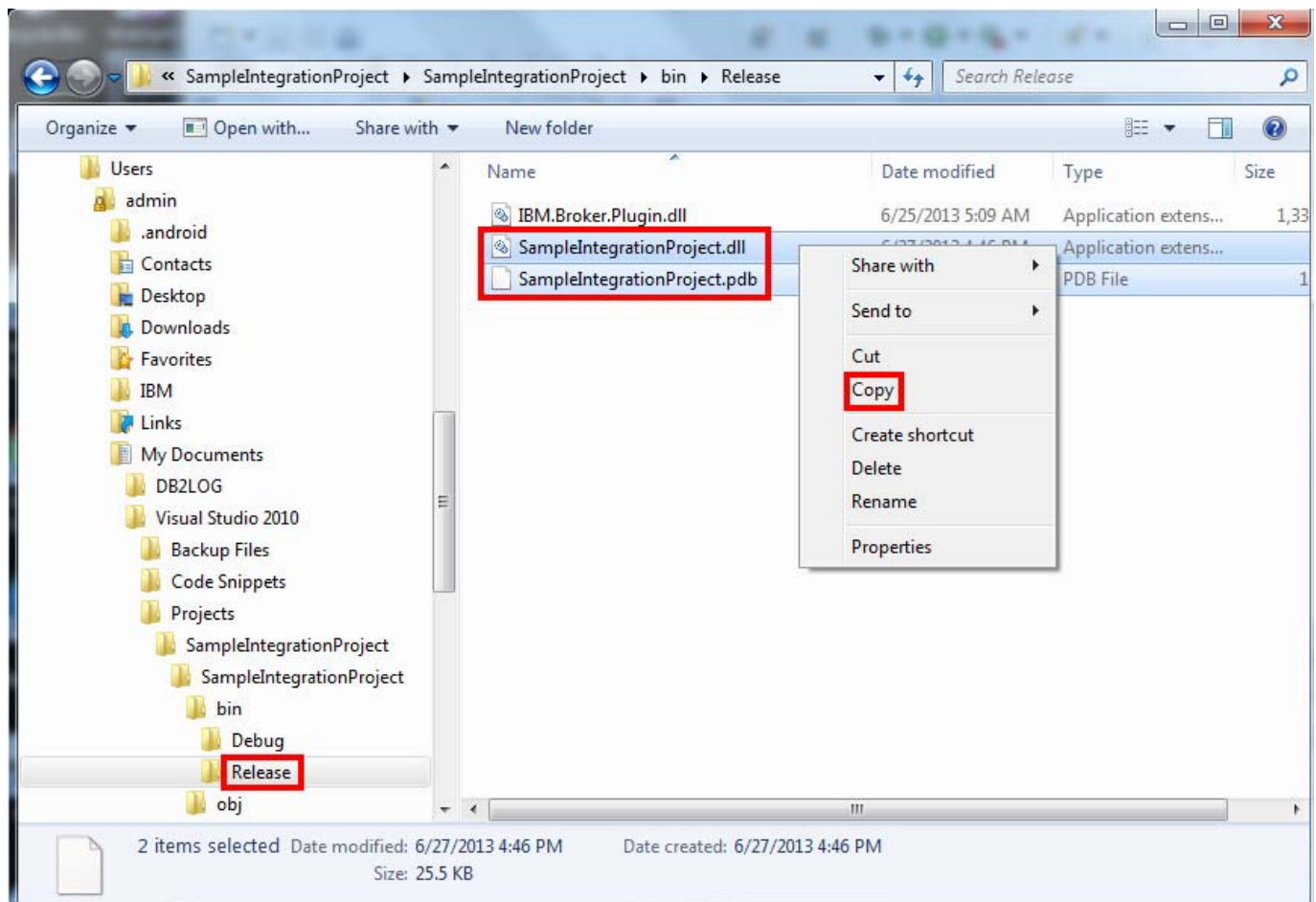
C:\users\admin\My Documents\Visual Studio 2010\Projects\SampleIntegrationProject\SampleIntegrationProject\bin\Release

__2. Select the **SampleIntegrationProject.dll**.

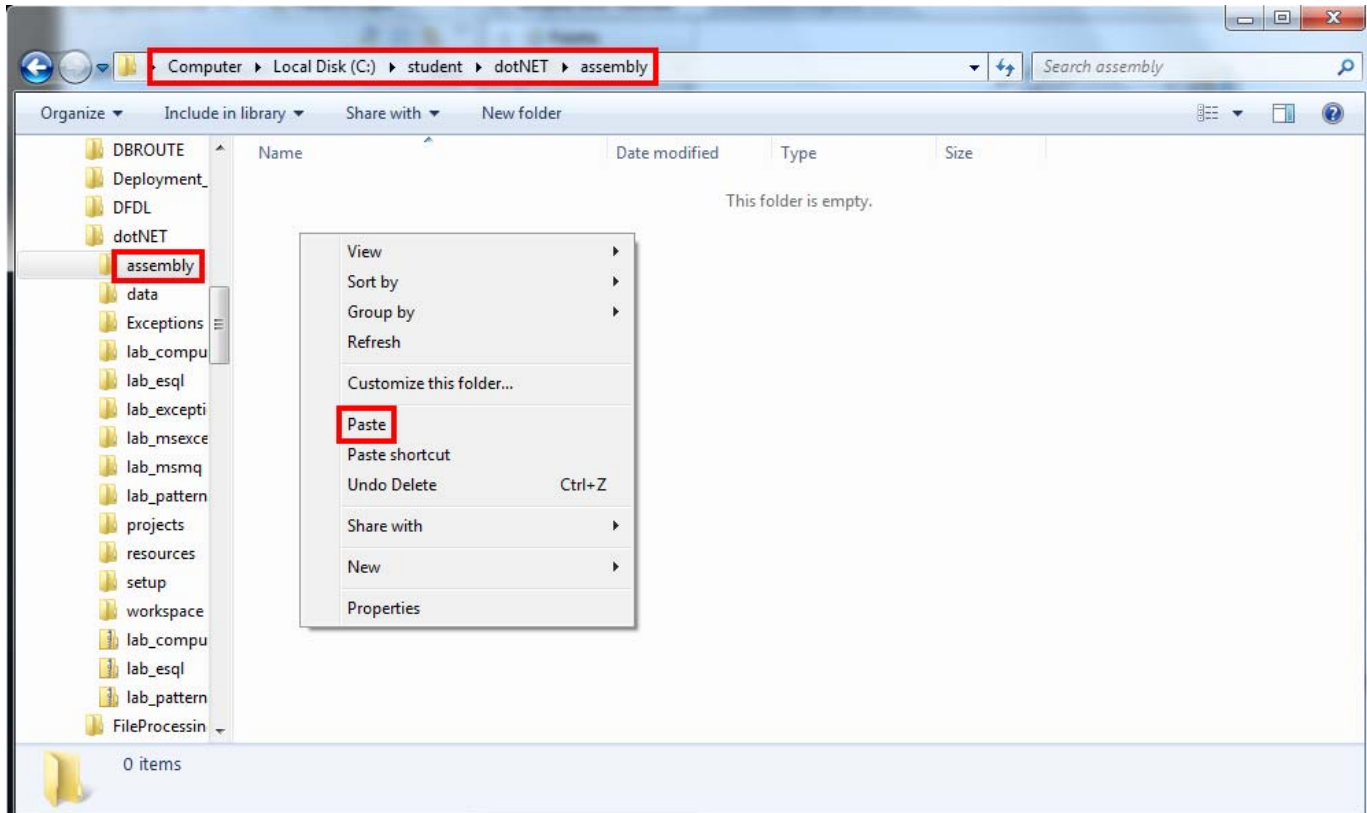
__3. Hold down Ctrl and select the **SampleIntegrationProject.pdb** files.

__4. Right click on one of the files.

__5. Select **Copy** from the menu.

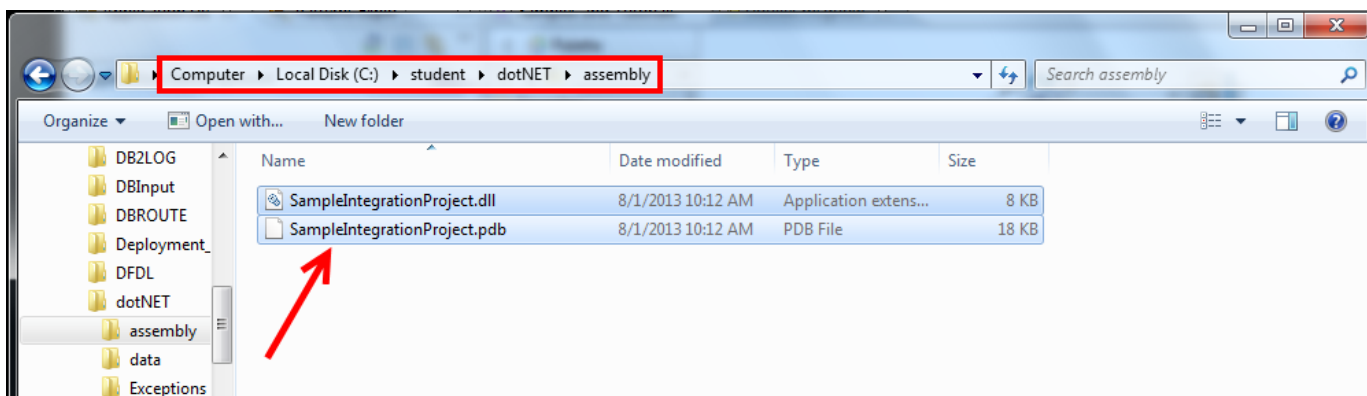


- __6. Navigate to the **C:\student\dotNET\assembly** directory.
- __7. Press the right mouse button.
- __8. Select **Paste** from the menu.

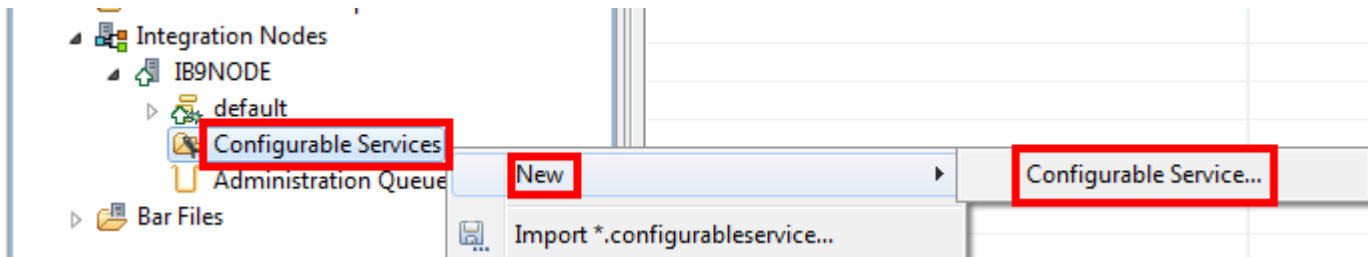


The two files should now be visible in the **assembly** directory.

- __9. Close the Windows Explorer session.



- __10. Switch to Integration (MQ) Explorer.
- __11. Expand **Integration Nodes->IB9NODE**.
- __12. Select **Configurable Services** under the **IB9NODE** node.
- __13. Press the right mouse button.
- __14. Select **New->Configurable Service**.



- __15. Enter **SampleDotNetConfigService** as the **Name**.
- __16. Use the drop-down menu to select **DotNetAppDomain** as the **Type**.
- __17. Enter **C:\student\dotNET\assembly** as the **ApplicationBase**.
- __18. Press the **Finish** button to create the configurable service.

Configurable Service
Create a new Configurable Service and set its attributes

*Name: SampleDotNetConfigService
*Type: DotNetAppDomain
Template: AppDomainTemplate

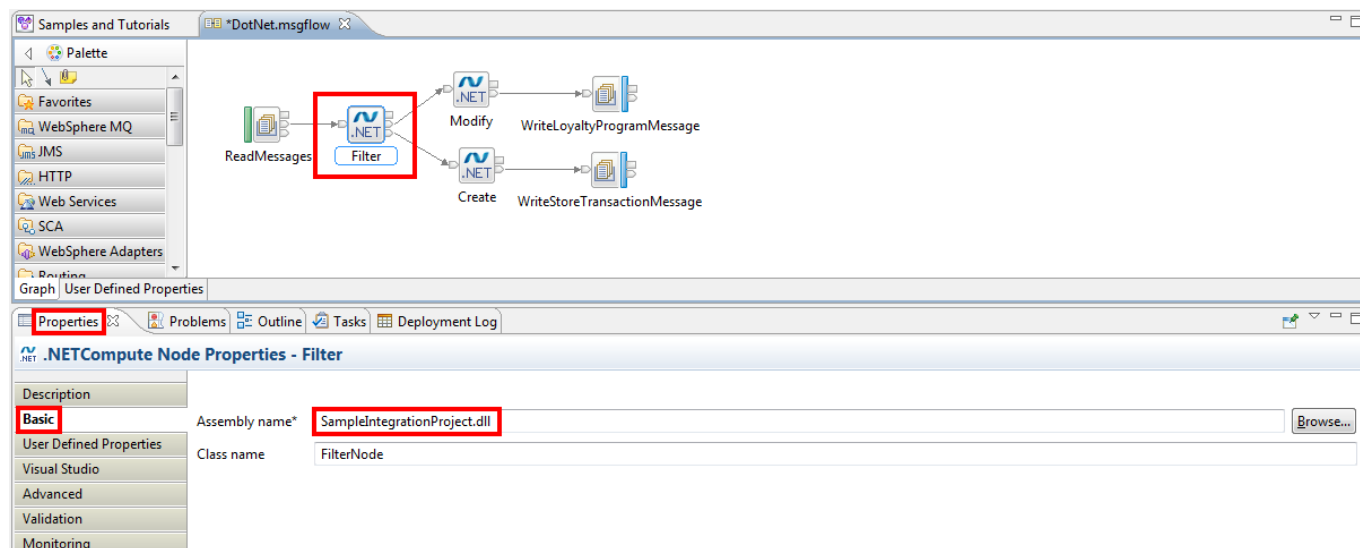
Key	Value
AllowHotSwapDeploy	true
ApplicationBase	C:\student\dotNET\assembly
ConfigurationFile	
DisallowCodeDownload	true
PrivateBinPath	
PrivateBinPathProbe	
ShadowCopyFiles	true
UseBrokerWorkpathForS...	false

Add Property Delete Property

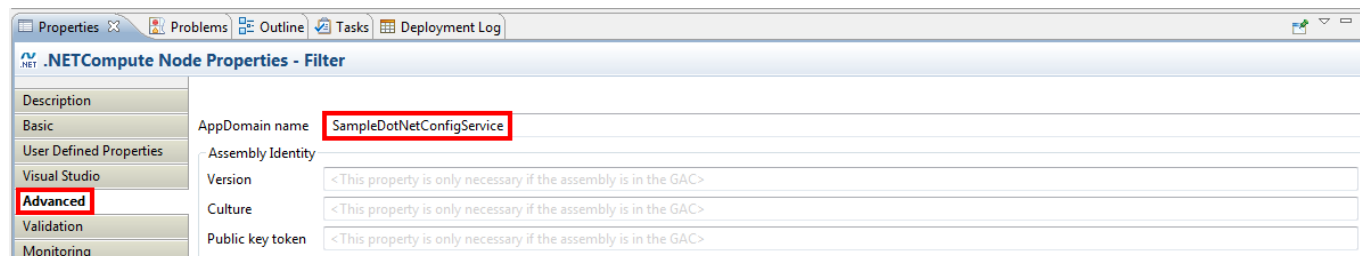
mqsicreateconfigurableservice IB9NODE -c DotNetAppDomain -o SampleDotNetConfigService -n AllowHotSwapDeploy,ApplicationBase,ConfigurationFile,DisallowCodeDownload,PrivateBinPath,PrivateBinPathProbe,ShadowCopyFiles,UseBrok

Finish Cancel

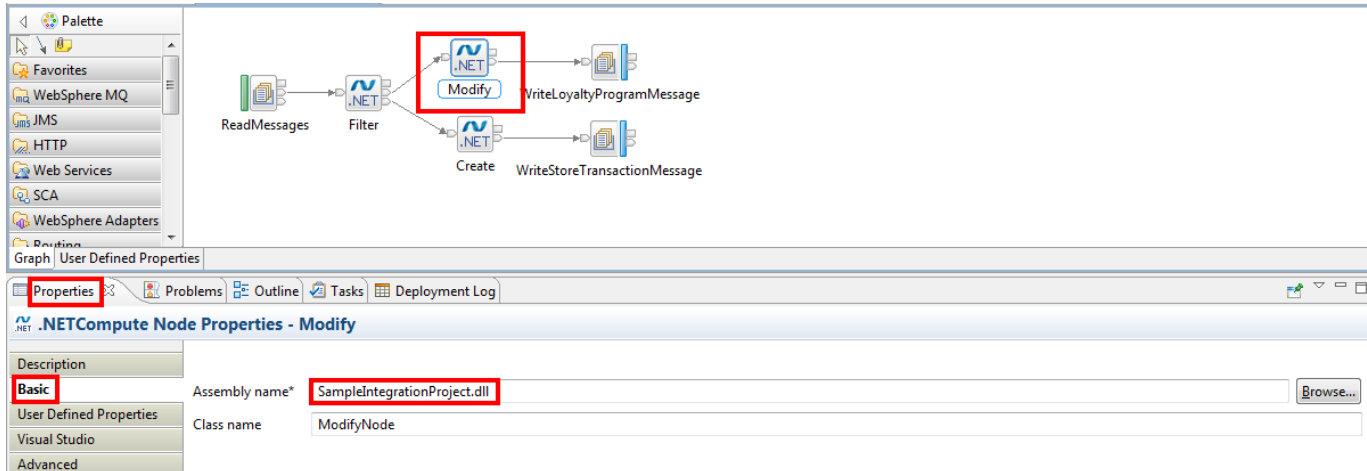
- __19. Return to the Integration Toolkit.
- __20. Select the **Filter** .NET compute node in the DotNet message flow.
- __21. In the **Properties** pane select the **Basic** tab.
- __22. Change the **Assembly name** to **SampleIntegrationProject.dll**.



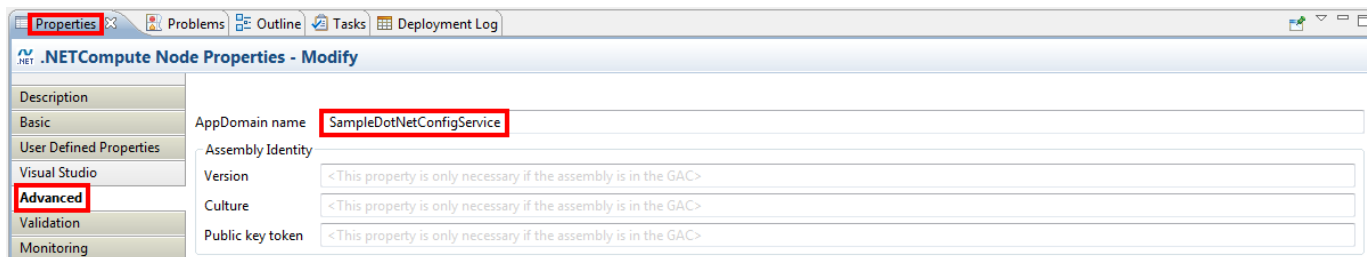
- __23. Select the **Advanced** tab.
- __24. Enter **SampleDotNetConfigService** as the **AppDomain name**.



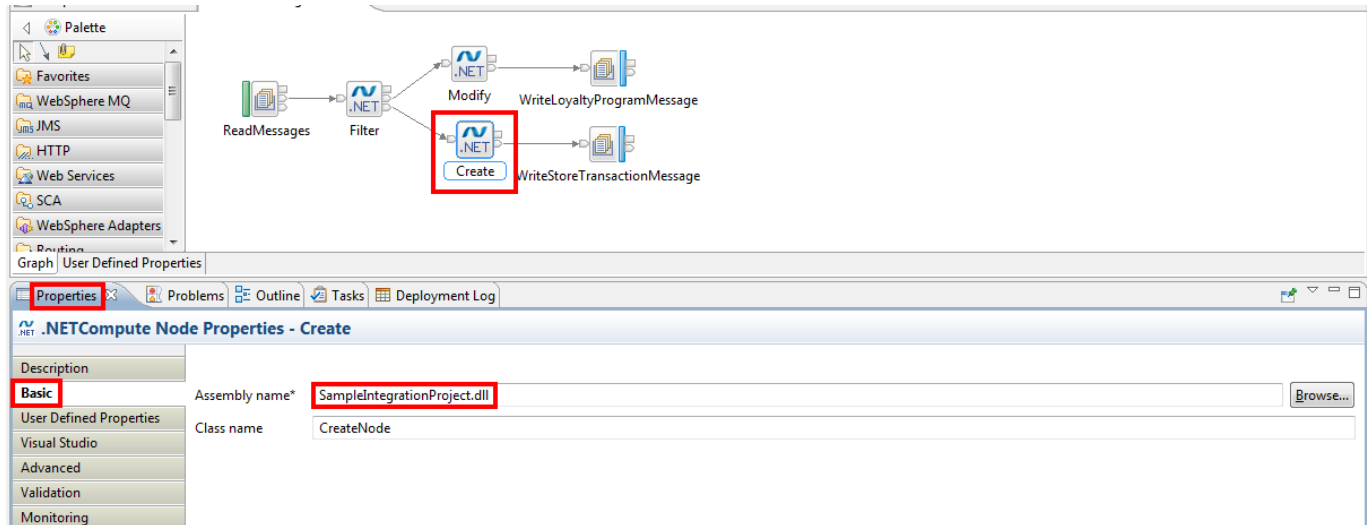
- __25. Select the **Modify** .NET compute node.
- __26. In the **Properties** pane select the **Basic** tab.
- __27. Change the **Assembly name** to **SampleIntegrationProject.dll**.



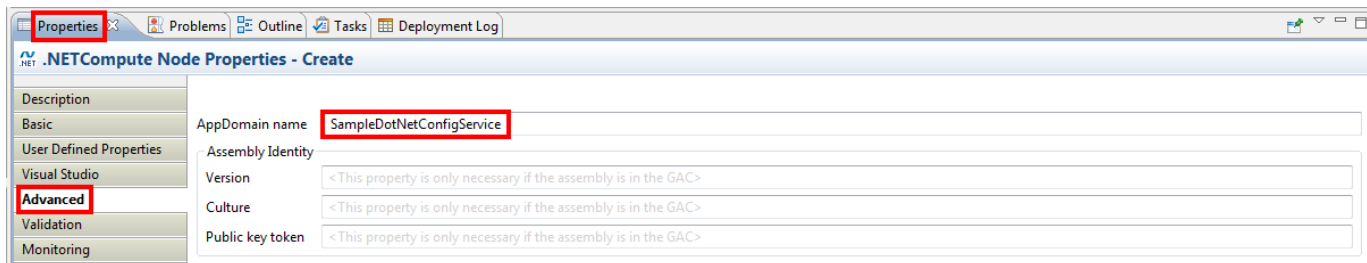
- __28. Select the **Advanced** tab.
- __29. Enter **SampleDotNetConfigService** as the **AppDomain name**.




- __30. Select the **Create** .NET compute node.
- __31. In the **Properties** pane select the **Basic** tab.
- __32. Change the **Assembly name** to **SampleIntegrationProject.dll**.



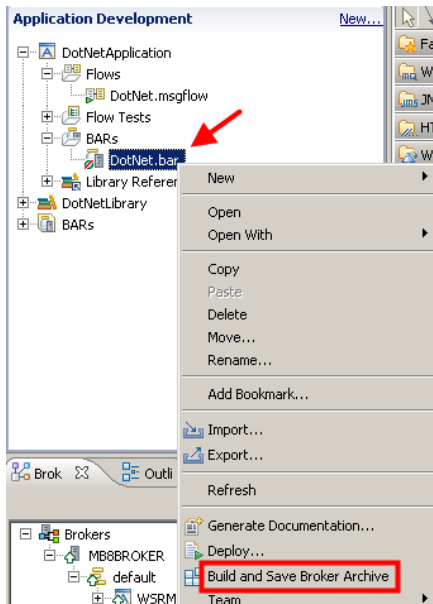
- __33. Select the **Advanced** tab.
- __34. Enter **SampleDotNetConfigService** as the **AppDomain name**.



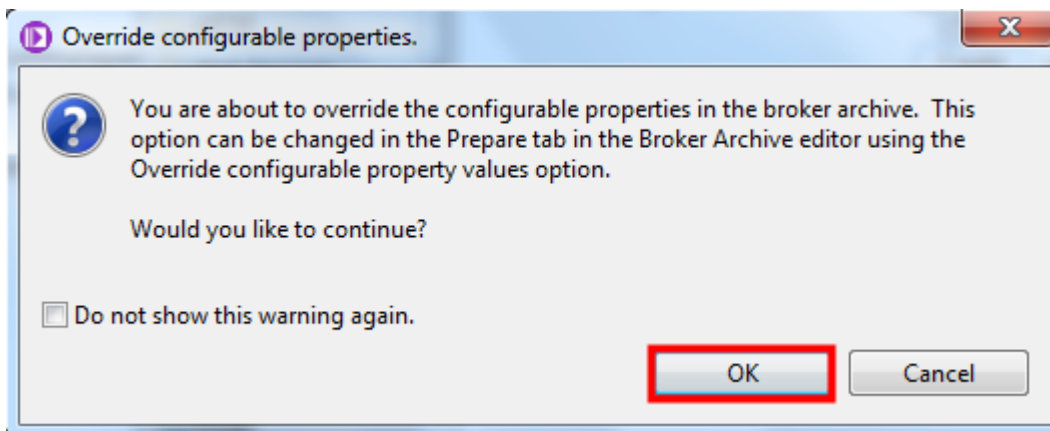
- __35.  Save the updated message flow (**Ctrl+S**).

5.5 Deploy and execute the message flow

- ___1. In the project navigator expand **DotNetApplication**→**BARs**.
- ___2. Select the **DotNet.bar** broker archive.
- ___3. Press the right mouse button.
- ___4. Select **Build and Save Broker Archive** from the menu.



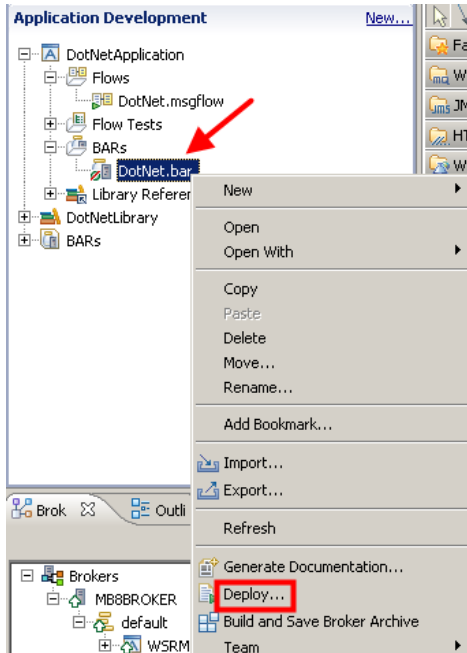
- ___5. Press the OK button to dismiss the warning dialog.



__6. Select the **DotNet.bar** broker archive.

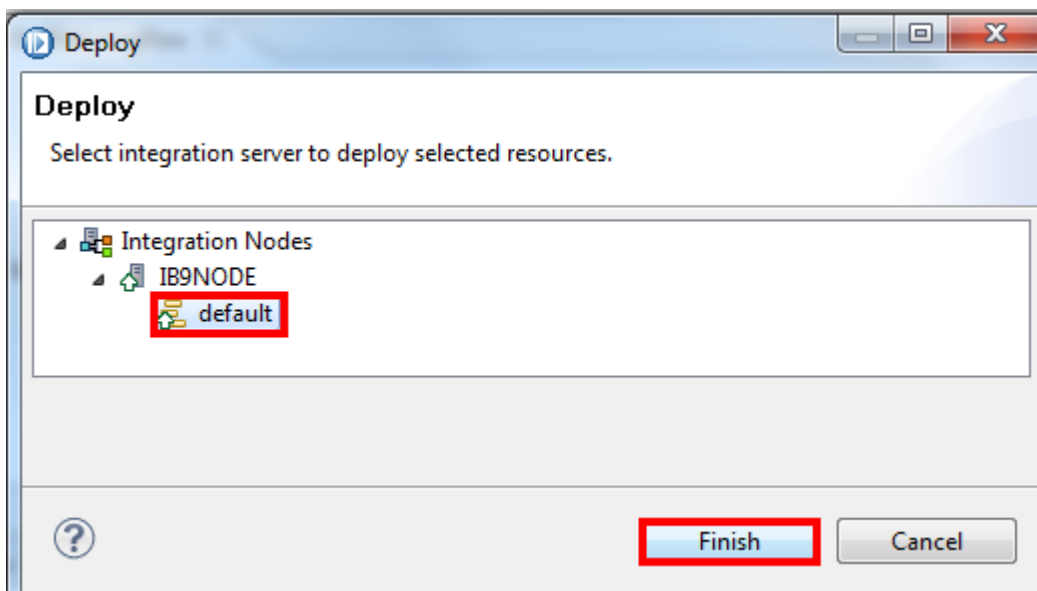
__7. Press the right mouse button.

__8. Select **Deploy** from the menu.

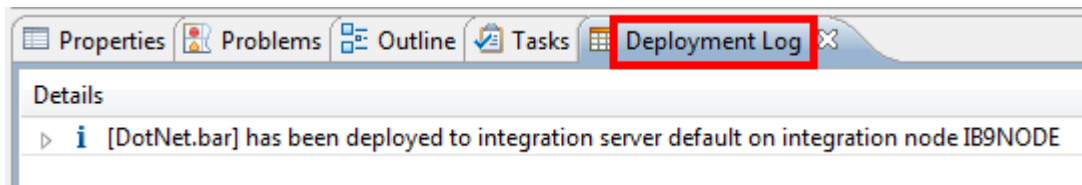


__9. Select the **default** integration server.

__10. Press the **Finish** button to start the deploy operation.

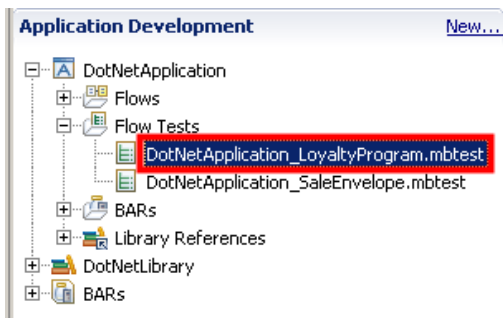


___11. Confirm that the deployment was successful.



___12. In the project navigator expand the **Flow Tests** folder.

___13. Double-click the **DotNetApplication_LoyaltyProgram.mbttest** test client file.



The input data is shown below.

```
<LoyaltyProgram
  xmlns:applicant="http://www.example.org/applicant"
  xmlns:store="http://www.example.org/store">
  <applicant:ApplicantDetails>
    <applicant:FirstName>Ben</applicant:FirstName>
    <applicant:LastName>Thompson</applicant:LastName>
    <applicant:HouseNo>1</applicant:HouseNo>
    <applicant:Street>Happiness Avenue</applicant:Street>
    <applicant:Town>Grumpyville</applicant:Town>
  </applicant:ApplicantDetails>
  <store:StoreDetails>
    <store:StoreID>001</store:StoreID>
  </store:StoreDetails>
</LoyaltyProgram>
```

- ___14. Select the **Configuration** tab.
- ___15. In the **Deployment location** section press the **Change** button.

The screenshot displays the IBM Integration Bus Configuration console. The left-hand pane shows a tree view under 'Test Client Configuration' with the following items: Message Flows, Deployment (selected), MQ Settings, JMS Settings, MQ Message Headers, MQ Message Header "Default Header", JMS Message Headers, and JMS Message Header "Default Header". The right-hand pane is titled 'Configuration' and contains several sections. At the top, there are two radio buttons: 'Only rebuild and deploy Broker Archive automatically when changes have occurred.' (selected) and 'Override configurable properties when rebuilding Broker Archive file.' Below this is section '2. Specify Broker Archive file' with a text area containing '/DotNetApplication/DotNet.bar' and 'Browse...' and 'Reset' buttons. Section '3. Deployment location' includes a 'Current Location:' label and a 'Change...' button (highlighted with a red box). Below this are input fields for 'Host' (localhost), 'Port' (empty), 'Broker' (IB9NODE), and 'Execution group' (DotNetExecutionGroup). At the bottom of this section are three checkboxes: 'Trace and debug' (unchecked), 'Stop at the beginning of the flow during debugging' (unchecked), and 'Always use the same deployment location for every test run' (checked). The bottom status bar shows 'Events' and 'Configuration' (highlighted with a red box).

Configuration

Test Client Configuration

- Message Flows
- Deployment**
- MQ Settings
- JMS Settings
- MQ Message Headers
 - MQ Message Header "Default Header"
- JMS Message Headers
 - JMS Message Header "Default Header"

☒ Only rebuild and deploy Broker Archive automatically when changes have occurred.
☐ Override configurable properties when rebuilding Broker Archive file.

2. Specify Broker Archive file

Specify the name of the Broker Archive file being deployed. If you have chosen manual deployment you must specify a file. If you have chosen automatic deployment you can optionally specify a file. If no file is specified, a system generated name will be assigned..

/DotNetApplication/DotNet.bar

3. Deployment location

Current Location:

Host: localhost

Port:

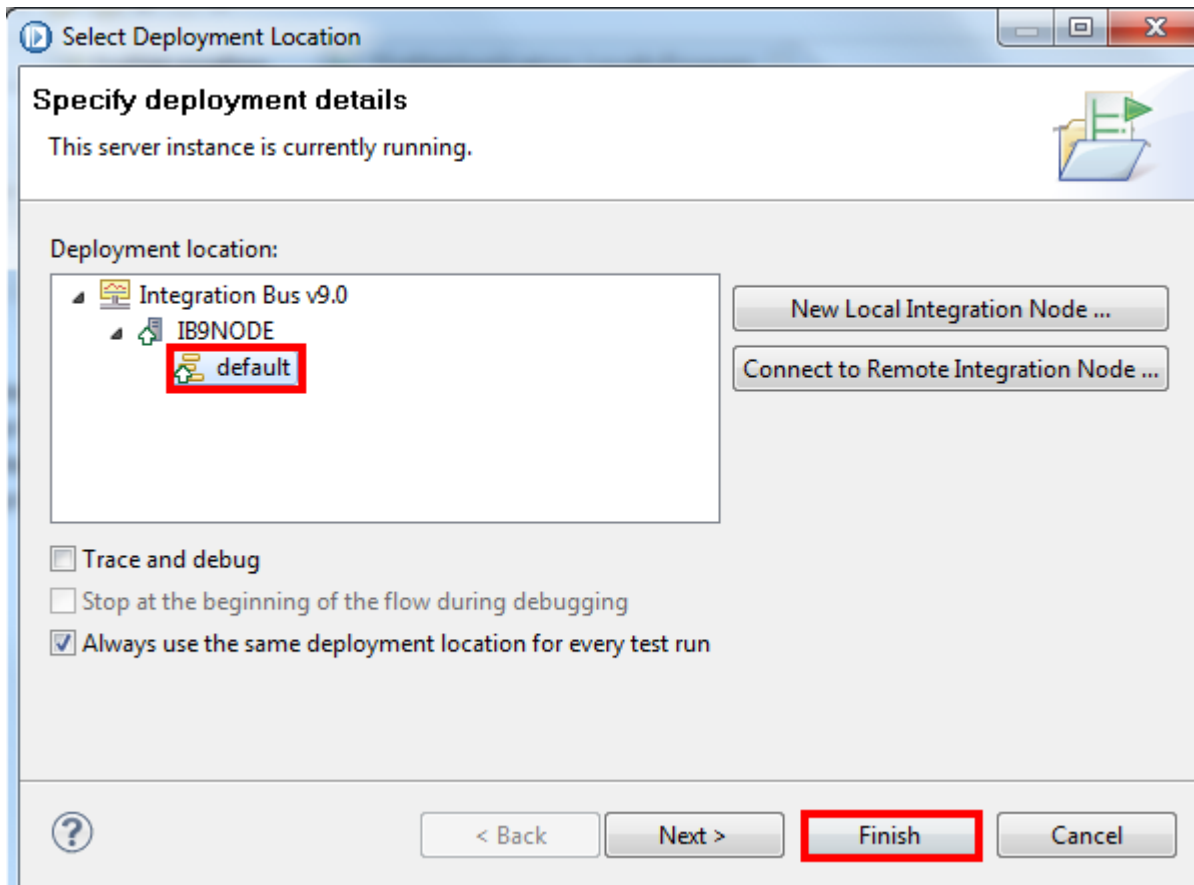
Broker: IB9NODE

Execution group: DotNetExecutionGroup

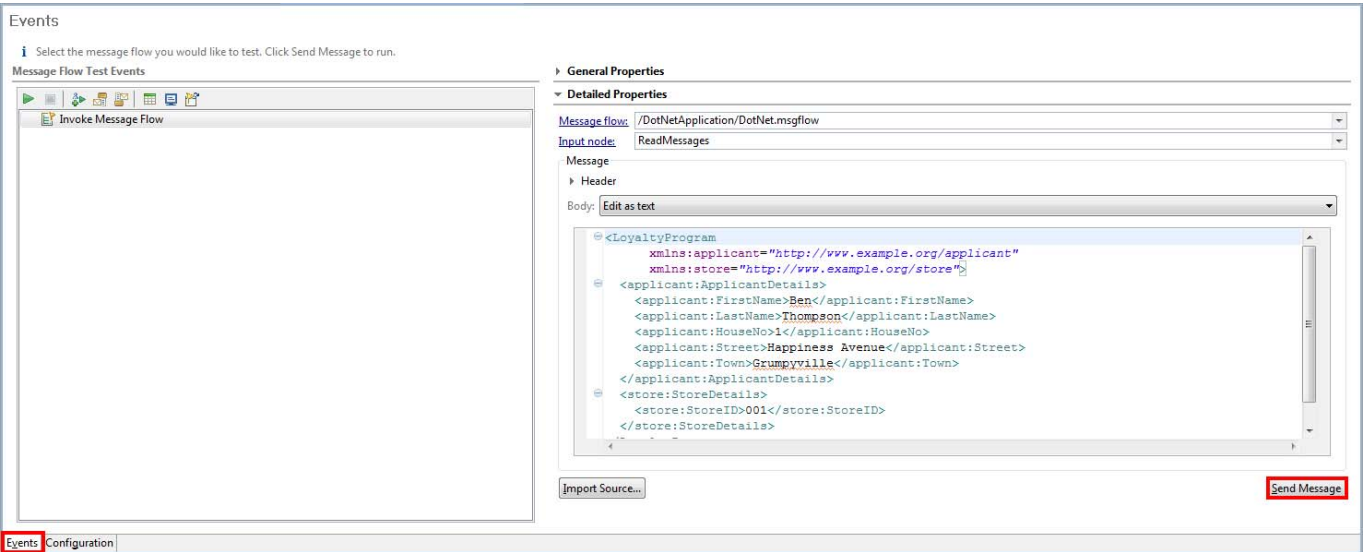
☐ Trace and debug
☐ Stop at the beginning of the flow during debugging
☒ Always use the same deployment location for every test run

Events **Configuration**

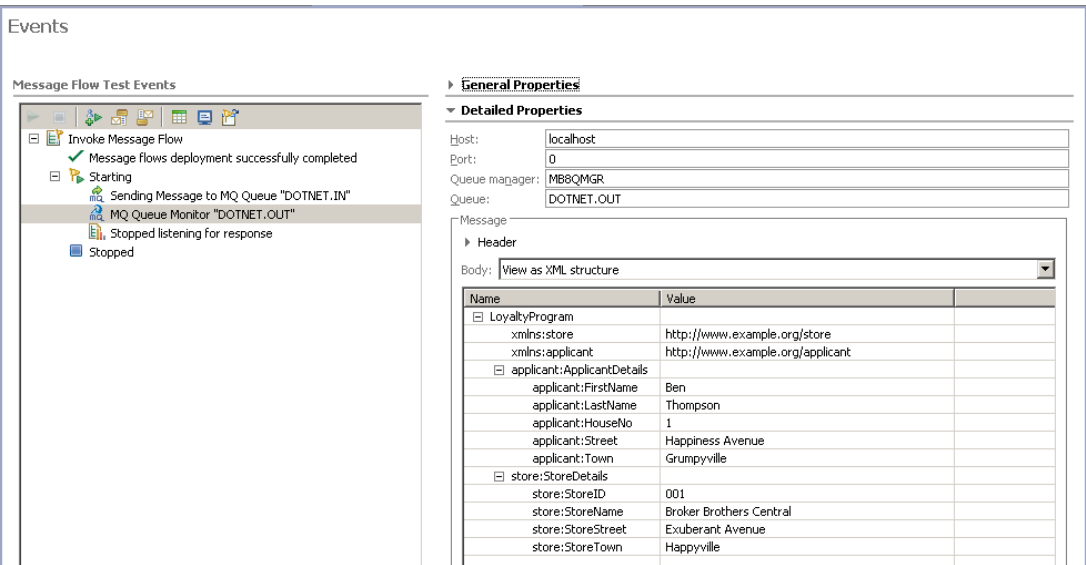
- __16. Select the **default** integration server.
- __17. Press the **Finish** button.



- ___18. Select the **Events** tab.
- ___19. Press the **Send Message** button.



- ___20. After the test has completed select the **MQ Queue Monitor “DOTNET.OUT”** item and review the result.

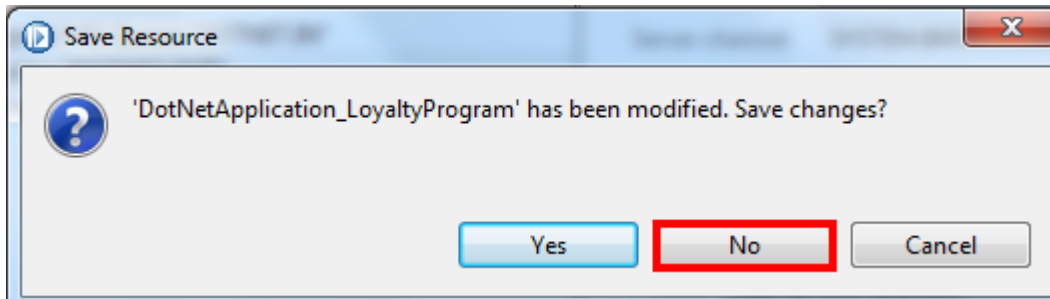


- ___21. Close the test client.



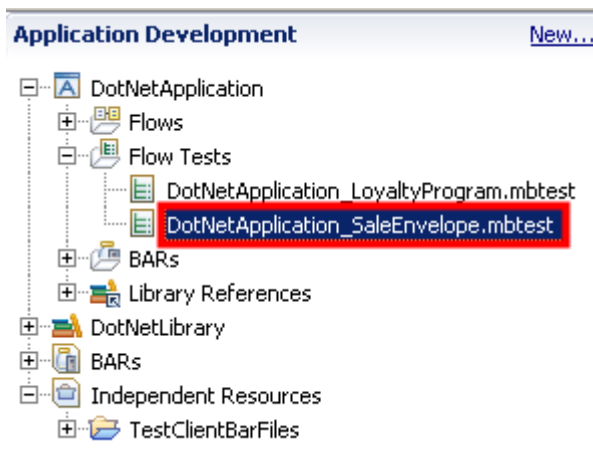
The test results will not be saved.

__22. Press the **No** button.



The other path through the message flow will be tested next.

__23. Double-click the **DotNetApplication_SaleEnvelope.mbstest** test client file in the **Flow Tests** folder.



24. The input data is shown below.

```

<SaleEnvelope>
  <Header>
    <SaleListCount>1</SaleListCount>
  </Header>
  <SaleList>
    <Invoice>
      <Initial>K</Initial>
      <Initial>A</Initial>
      <Surname>Braithwaite</Surname>
      <Item>
        <Code>00</Code>
        <Code>01</Code>
        <Code>02</Code>
        <Description>Twister</Description>
        <Category>Games</Category>
        <Price>00.30</Price>
        <Quantity>01</Quantity>
      </Item>
      <Item>
        <Code>02</Code>
        <Code>03</Code>
        <Code>01</Code>
        <Description>The Times Newspaper</Description>
        <Category>Books and Media</Category>
        <Price>00.20</Price>
        <Quantity>01</Quantity>
      </Item>
      <Balance>00.50</Balance>
      <Currency>Sterling</Currency>
    </Invoice>
    <Invoice>
      <Initial>T</Initial>
      <Initial>J</Initial>
      <Surname>Dunnwin</Surname>
      <Item>
        <Code>04</Code>
        <Code>05</Code>
        <Code>01</Code>
        <Description>The Origin of Species</Description>
        <Category>Books and Media</Category>
        <Price>22.34</Price>
        <Quantity>02</Quantity>
      </Item>
      <Item>
        <Code>06</Code>
        <Code>07</Code>
        <Code>01</Code>
        <Description>Microscope</Description>
        <Category>Miscellaneous</Category>
        <Price>36.20</Price>
        <Quantity>01</Quantity>
      </Item>
      <Balance>81.84</Balance>
      <Currency>Euros</Currency>
    </Invoice>
  </SaleList>
  <Trailer>
    <CompletionTime>12.00.00</CompletionTime>
  </Trailer>
</SaleEnvelope>

```


- ___25. Select the **Configuration** tab.
- ___26. In the **Deployment location** section press the **Change** button.

The screenshot shows the IBM Integration Bus Configuration console. The left pane displays a tree view under 'Test Client Configuration' with 'Deployment' selected. The right pane shows configuration options for the selected deployment. At the bottom, the 'Configuration' tab is highlighted in the console's tab bar.

Configuration

Test Client Configuration

- Message Flows
- Deployment**
- MQ Settings
- JMS Settings
 - MQ Message Headers
 - MQ Message Header "Default Header"
 - JMS Message Headers
 - JMS Message Header "Default Header"

☒ Only rebuild and deploy Broker Archive automatically when changes have occurred.
☐ Override configurable properties when rebuilding Broker Archive file.

2. Specify Broker Archive file

Specify the name of the Broker Archive file being deployed. If you have chosen manual deployment you must specify a file. If you have chosen automatic deployment you can optionally specify a file. If no file is specified, a system generated name will be assigned..

/DotNetApplication/DotNet.bar Browse... Reset

3. Deployment location

Current Location: Change...

Host: localhost

Port:

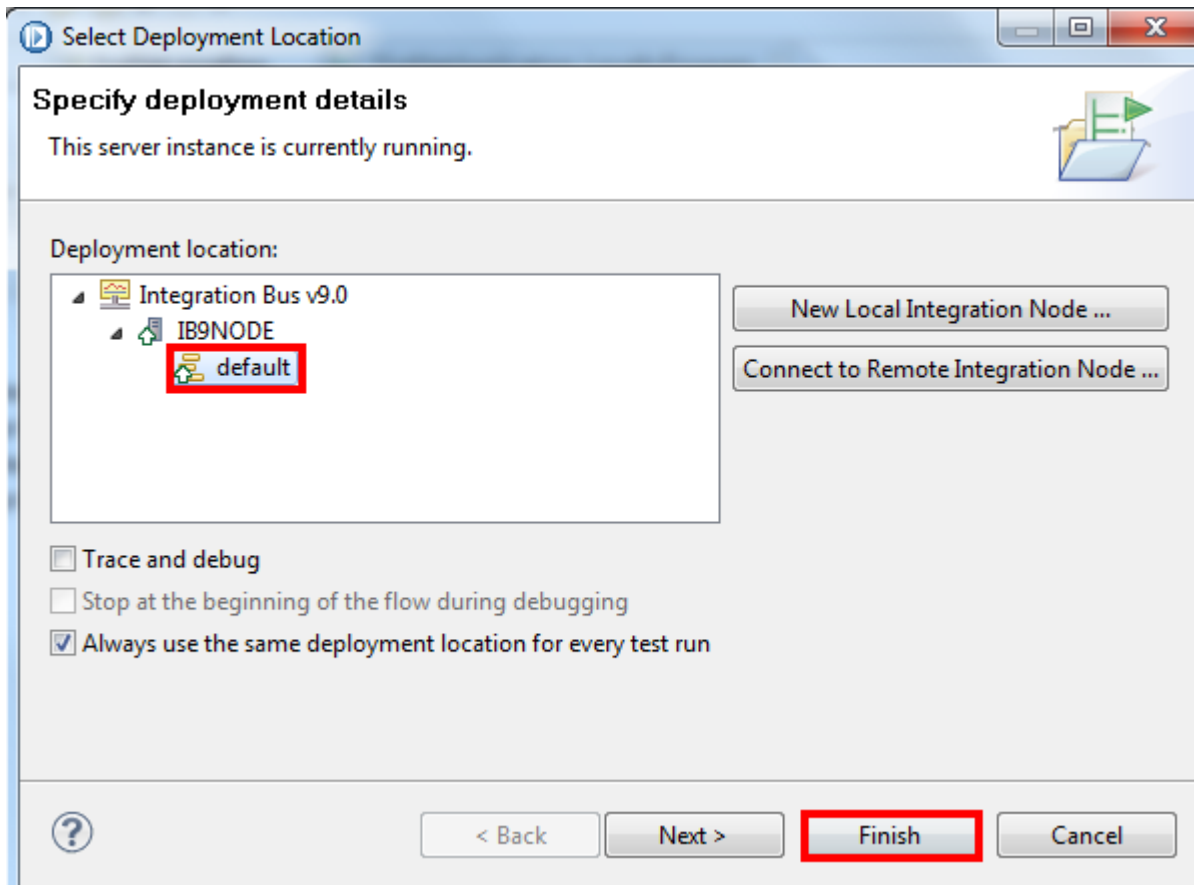
Broker: IB9NODE

Execution group: DotNetExecutionGroup

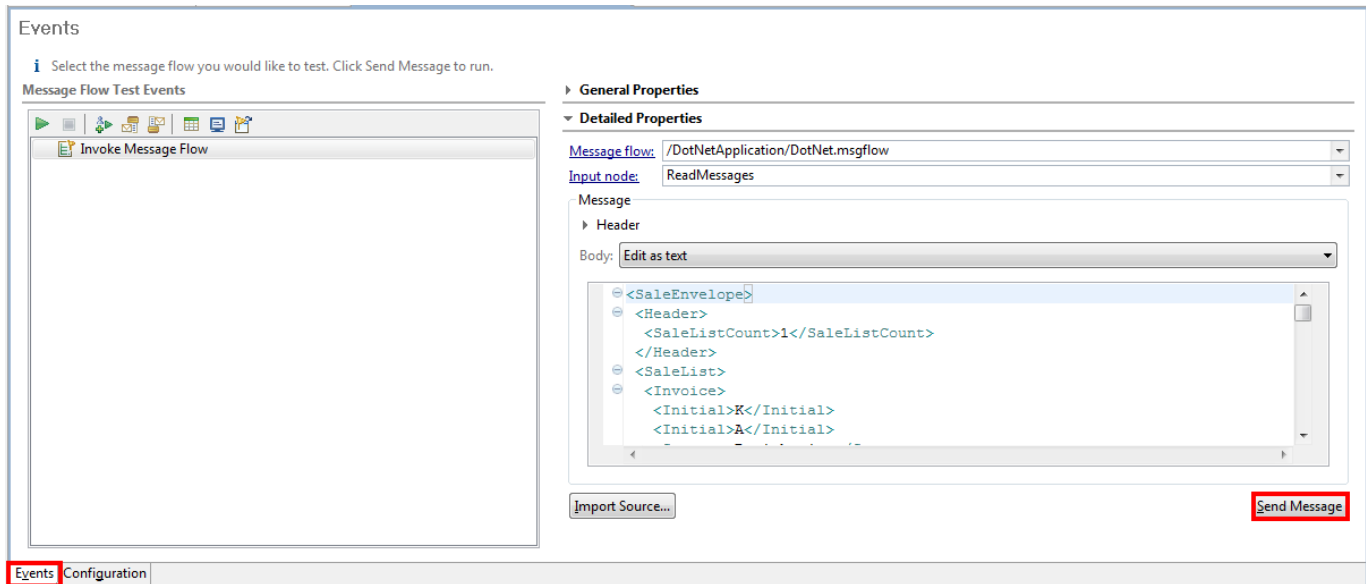
☐ Trace and debug
☐ Stop at the beginning of the flow during debugging
☒ Always use the same deployment location for every test run

Events **Configuration**

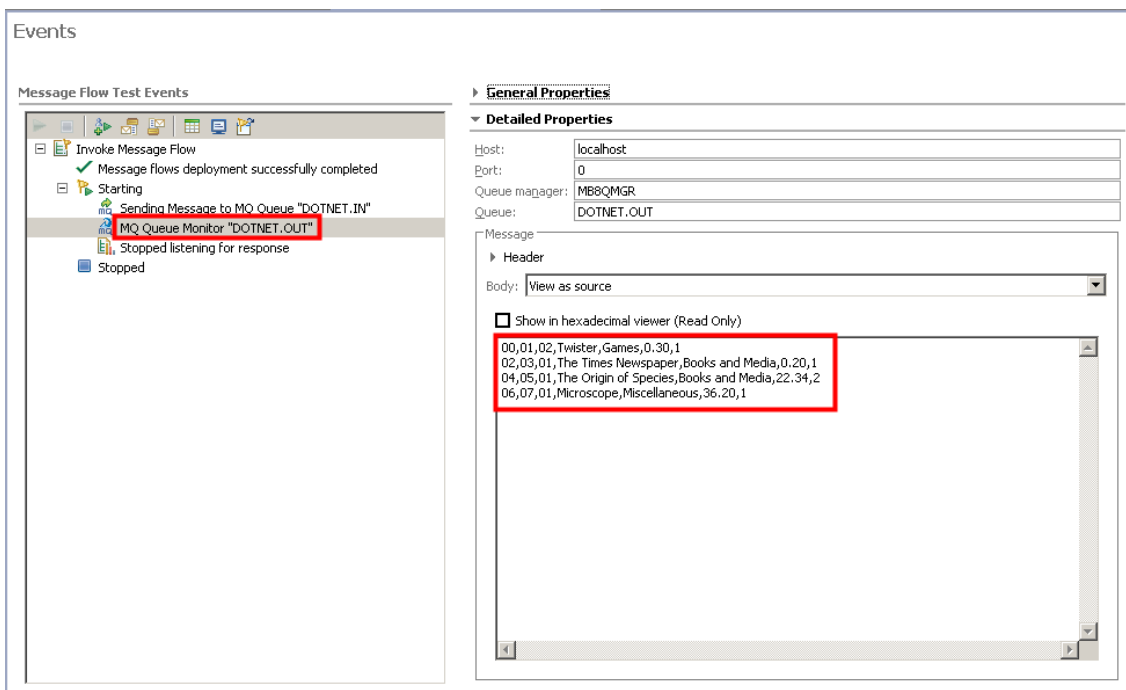
- __27. Select the **default** execution group.
- __28. Press the **Finish** button.



- __29. Select the **Events** tab.
- __30. Press the **Send Message** button.



- __31. After the test has completed select the **MQ Queue Monitor "DOTNET.OUT"** item and review the result.

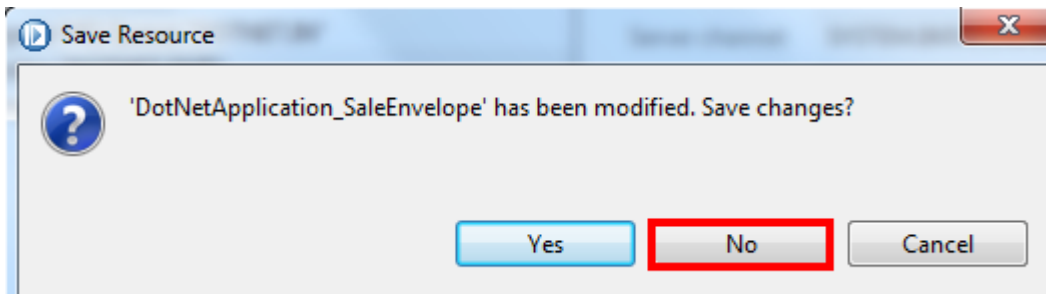


__32. Close the test client.



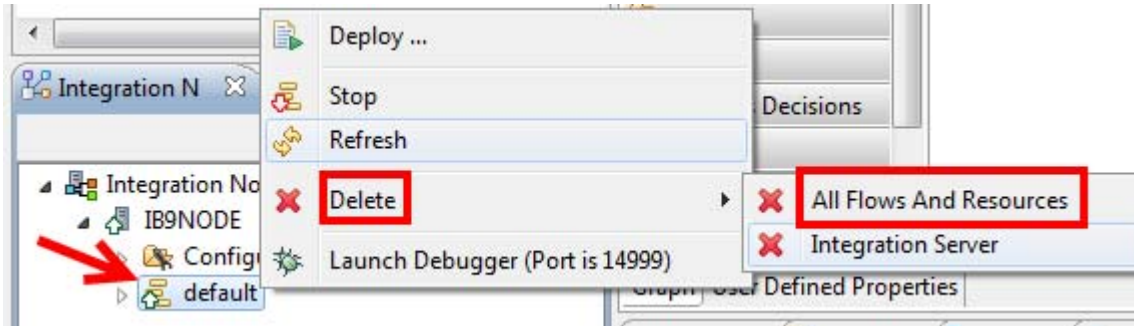
The test results will not be saved.

__33. Press the **No** button.

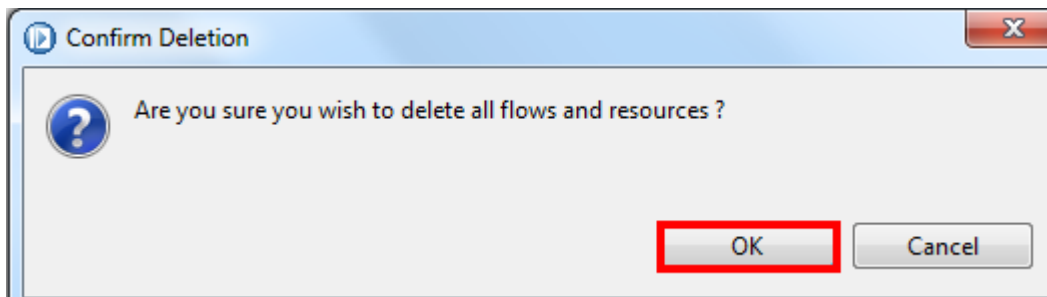


5.6 Cleanup

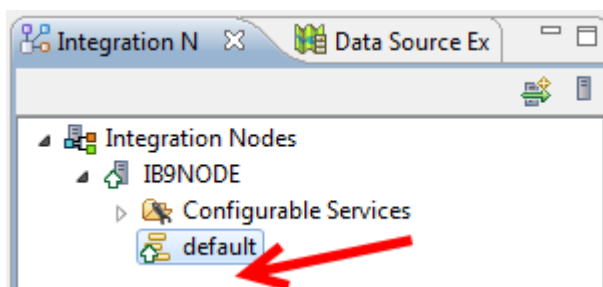
- ___1. In the **Integration Nodes** pane expand the **IB9NODE** entry.
- ___2. Select the **default** integration server.
- ___3. Press the right mouse button.
- ___4. Select **Delete**→**All Flows and Resources** from the menu.



- ___5. Press the **OK** button to acknowledge the warning and start the deletion.



The **default** integration server should now be empty.



This is the end of Lab 5.

Lab 6 Working with mobile applications

6.1 Overview

IBM Integration Bus Version 9.0 contains four patterns that facilitate integration with mobile applications, through the Worklight product suite. This lab also makes use of the global caching function in Integration Bus, which has been activated in the VMWare image. The four patterns are:

- **Service, management**

The service management pattern integrates a mobile application written for the Worklight platform with a service running in IBM Integration Bus. You can use the pattern to make an Integration Bus service available through REST APIs invoked by mobile applications running on all types of devices.

- **Resource handler**

This pattern provides services to mobile applications that use the Worklight APIs. The services are exposed to mobile applications as Worklight adapter procedures that are invoked from JavaScript in the application.

The pattern is customized by implementing handlers for each of these procedures as sub flows in IBM Integration Bus. The pattern provides security policy enforcement and global caching. The pattern uses REST calls passing data in a JSON format to the generated message flows.

- **Microsoft .Net request and response**

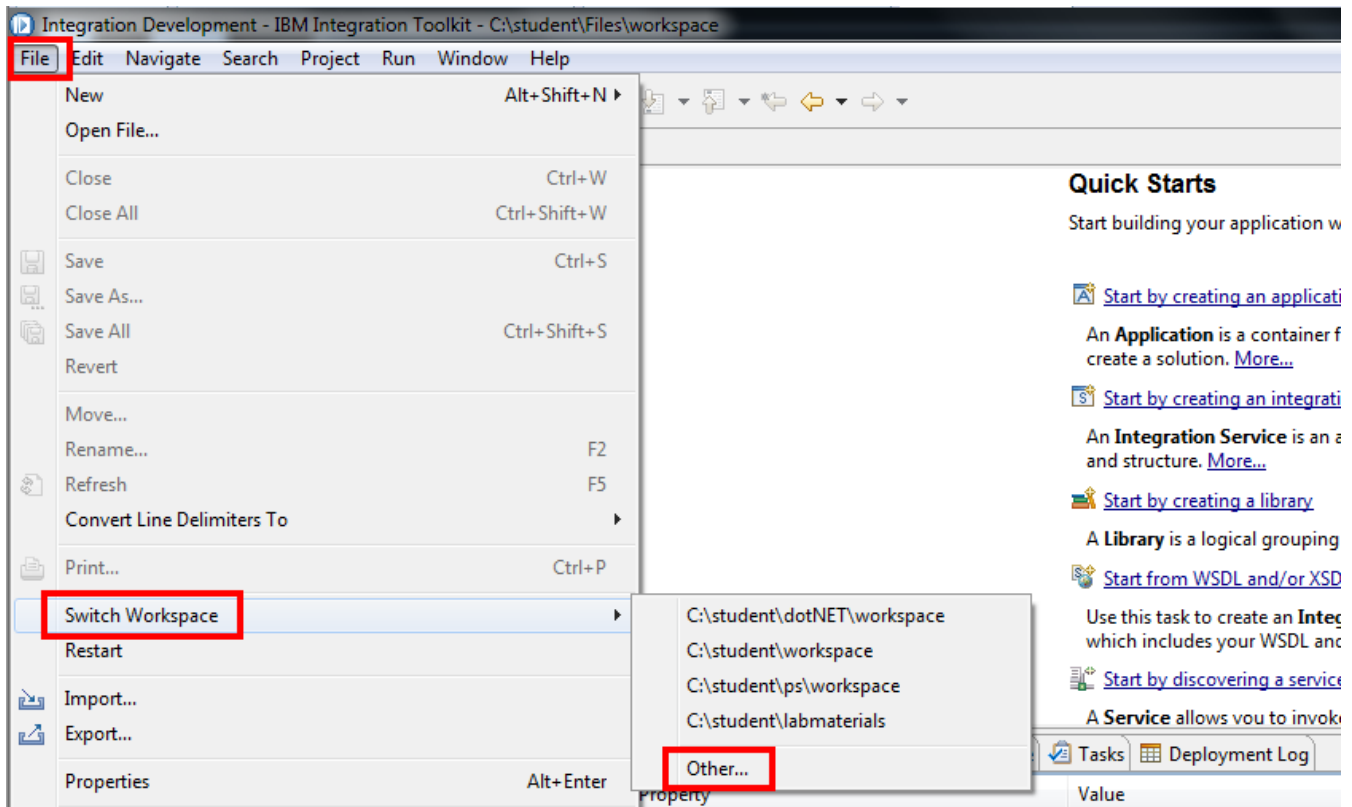
This pattern integrates a mobile application written for the Worklight platform with Microsoft .NET applications. This pattern makes a Microsoft .NET class available through REST APIs invoked by mobile applications running on all types of devices.

- **Worklight Integration from MQ**

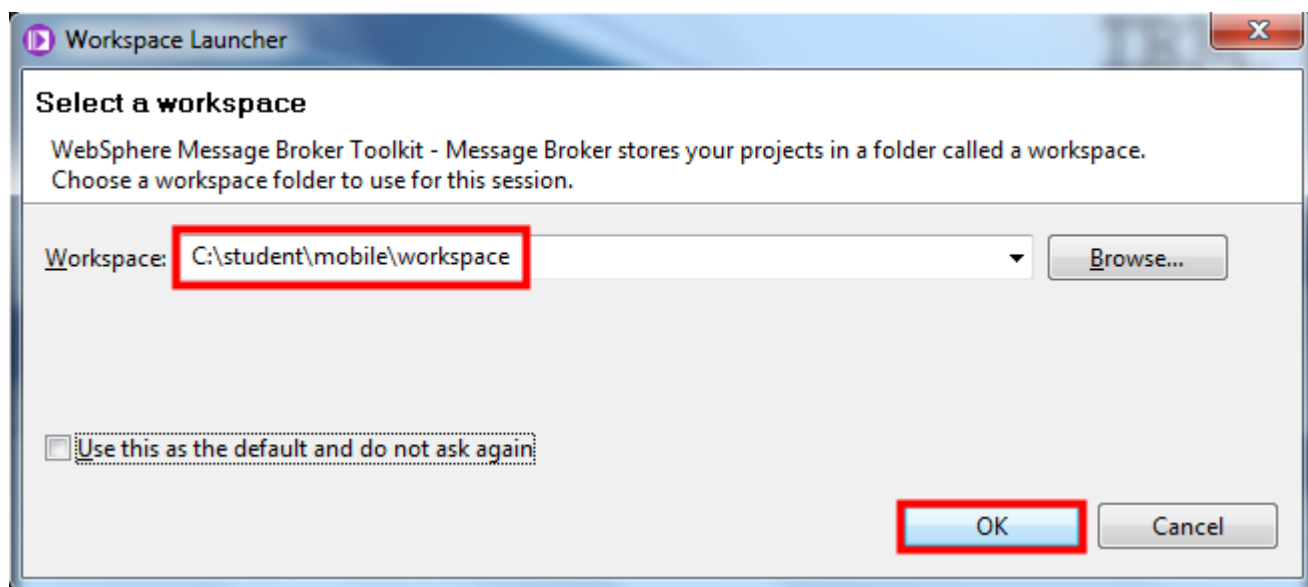
The Worklight push notification from WebSphere MQ pattern sends notifications to Worklight mobile applications using WebSphere MQ.

6.2 Microsoft .NET request/response mobile pattern

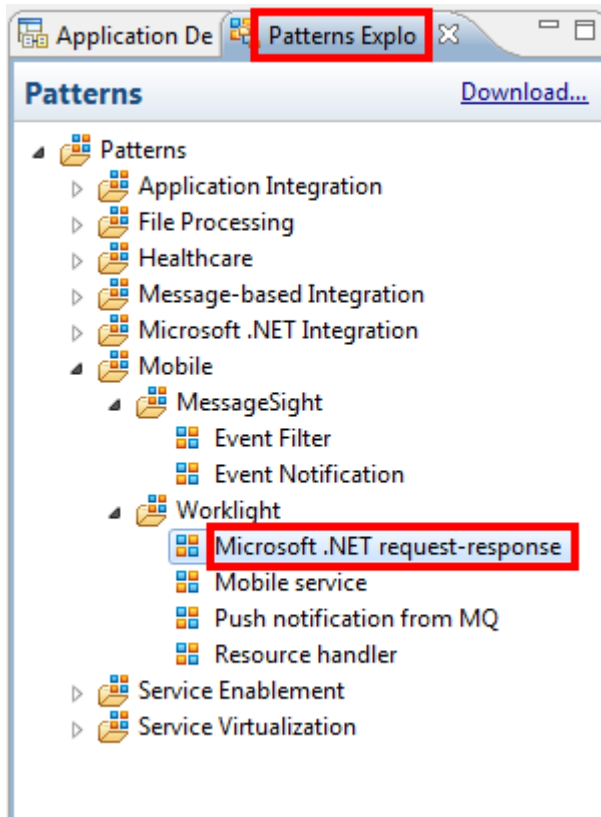
- __1. Return to the Integration Toolkit.
- __2. Select **File**→**Switch Workspace**→**Other**.



- ___3. Enter **C:\student\mobile\workspace** as the **Workspace** (or use the **Browse** button).
- ___4. Press the **OK** button to continue.

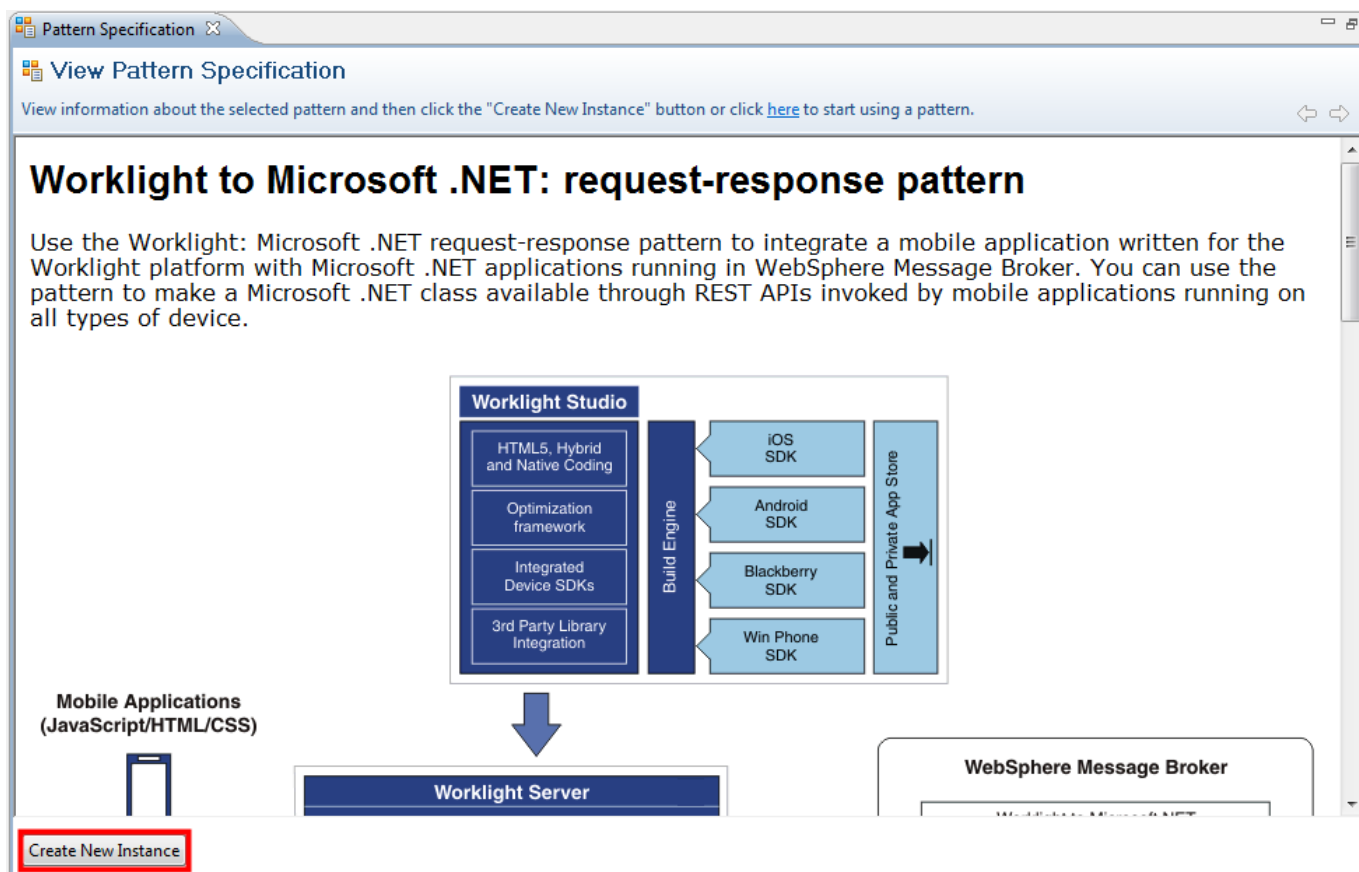


- __5. In the Integration Toolkit, select the **Patterns Explorer**.
- __6. If necessary, expand the **Mobile→Worklight** categories.
- __7. Select the **Microsoft .Net request/response** pattern.



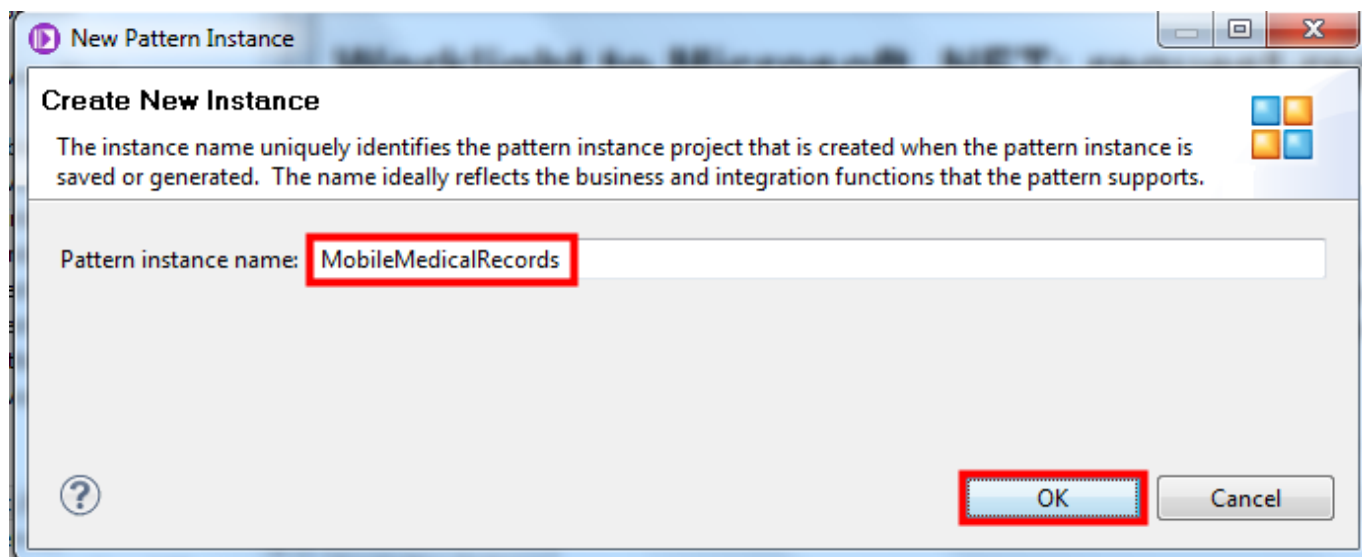
Take some time to read about the pattern.

__8. Press the **Create New Instance** button.



__9. Set the **Pattern instance name** to **MobileMedicalRecords**.

__10. Click **OK**.



- __11. Expand the **Worklight** section.
- __12. Remove the selection from the **Enable audit** check box. Audit is not configured on the image and this will result in a null pointer run time exception if it is selected.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

⚠ Configure the .NET assembly that the service invokes.

Pattern Parameters

Worklight (Expanded)

Configure the Worklight integration adapter

Worklight version: Worklight v5.0

Adapter description: Worklight integration adapter

Maximum concurrent connections *: 99

Enable audit *: ☒

Microsoft .NET assembly (Collapsed)

Service information (Collapsed)

Logging (Collapsed)

Error handling (Collapsed)

General (Collapsed)

Pattern Parameters Details

- Worklight
- Microsoft .NET assembly
- Service information
- Logging
- Error handling
- General

- __13. Collapse the **Worklight** section.
- __14. Expand the **Microsoft .NET assembly** section.
- __15. Press the **Configure** button.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

⚠ Configure the .NET assembly that the service invokes.

Pattern Parameters

Worklight (Collapsed)

Microsoft .NET assembly (Expanded)

Configure the .NET assembly that implements the service calls

Class name: **Configure...**

Service information (Expanded) ✓

Logging (Expanded) ✓

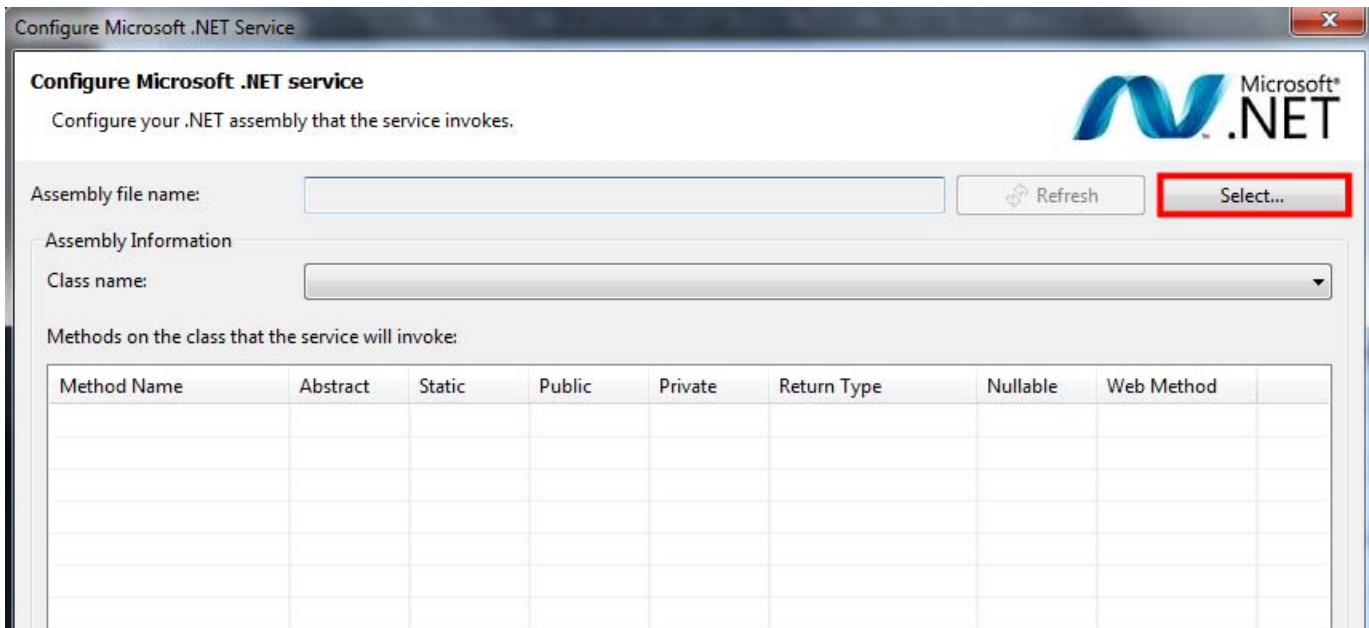
Error handling (Expanded) ✓

General (Expanded) ✓

Pattern Parameters Details

- Worklight
- Microsoft .NET assembly
- Service information
- Logging
- Error handling
- General

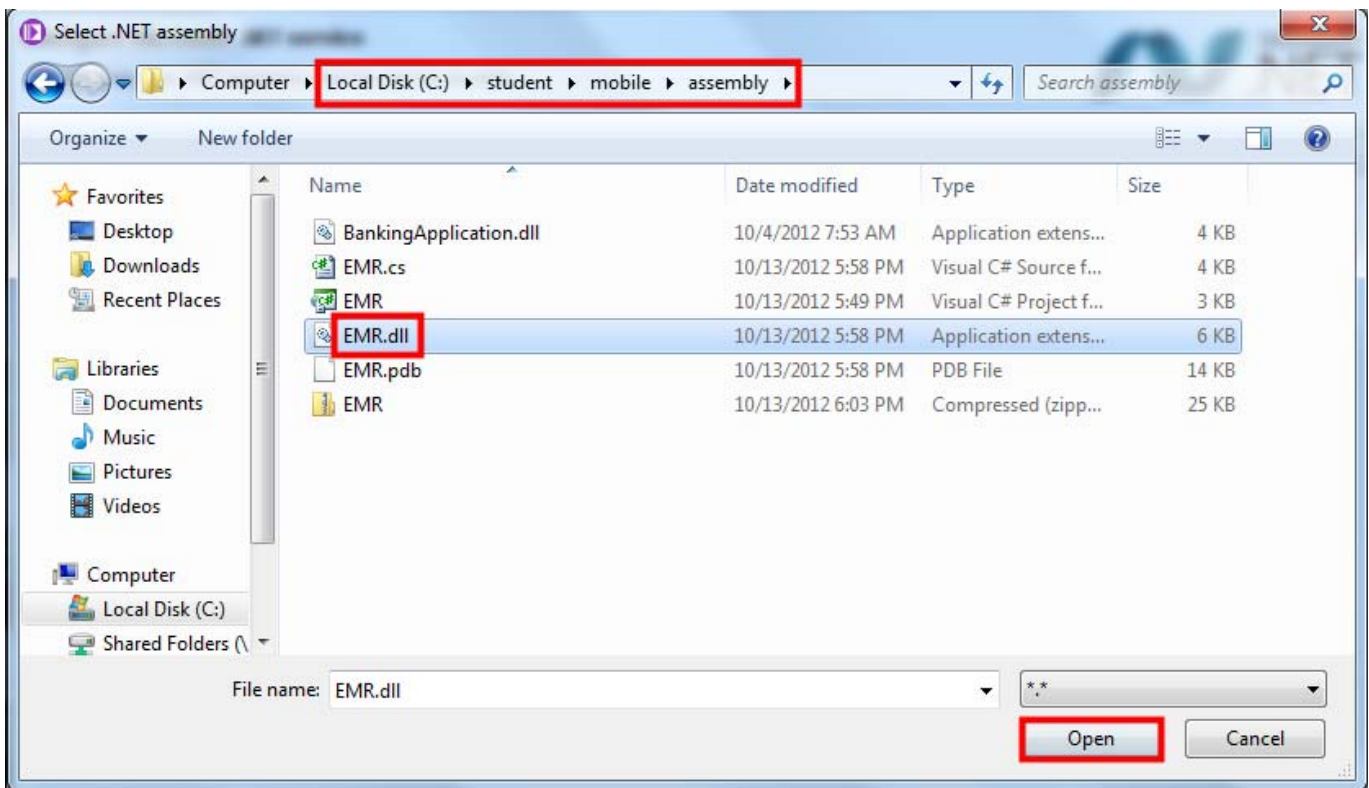
__16. Press the **Select** button.



__17. Navigate to the **C:\student\mobile\assembly** folder.

__18. Select the **EMR.dll** assembly.

__19. Press the **Open** button.



Notice that six methods have been extracted from the .NET assembly file. All the extracted methods have been selected.

__20. Accept the default selections.

__21. Press the **OK** button.

Configure Microsoft .NET service

Configure your .NET assembly that the service invokes.

Assembly file name:

Assembly Information

Class name:

Methods on the class that the service will invoke:

Method Name	Abstract	Static	Public	Private	Return Type	Nullable	Web Method
<input checked="" type="checkbox"/> GetPatientHistory	No	Yes	Yes	No	System.String	No	No
<input checked="" type="checkbox"/> FindPatientMRN	No	Yes	Yes	No	System.String	No	No
<input checked="" type="checkbox"/> GetVitalSigns	No	Yes	Yes	No	System.Void	No	No
<input checked="" type="checkbox"/> GetMedications	No	Yes	Yes	No	System.String	No	No
<input checked="" type="checkbox"/> GetLabResults	No	Yes	Yes	No	System.String	No	No
<input checked="" type="checkbox"/> RequestTest	No	Yes	Yes	No	System.Void	No	No
<input type="checkbox"/> ToString	No	No	Yes	No	System.String	No	No
<input type="checkbox"/> Equals	No	No	Yes	No	System.Boolean	No	No
<input type="checkbox"/> GetHashCode	No	No	Yes	No	System.Int32	No	No
<input type="checkbox"/> GetType	No	No	Yes	No	System.Type	No	No

☒ Select All ☐ Clear All

Parameters:

Parameter Name	Type	Input	Output	Reference	Optional	Nullable
MRN	System.String	Yes	No	No	No	No
admitted	System.Boolean	No	Yes	No	No	No
lastAdmitDate	System.String	No	Yes	No	No	No

The selection should now be shown in the **Microsoft .NET assembly** configuration.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern Parameters

- Worklight
- Microsoft .NET assembly**
 - Configure the .NET assembly that implements the service calls
 - Class name: EMR
 - Configure...
- Service information
- Logging
- Error handling
- General

Pattern Parameters Details

- Worklight
- Microsoft .NET assembly
- Service information
- Logging
- Error handling
- General

__22. Collapse the **Microsoft .NET assembly** section.

__23. Expand the **Logging** section.

__24. Select the **Logging required** check box.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

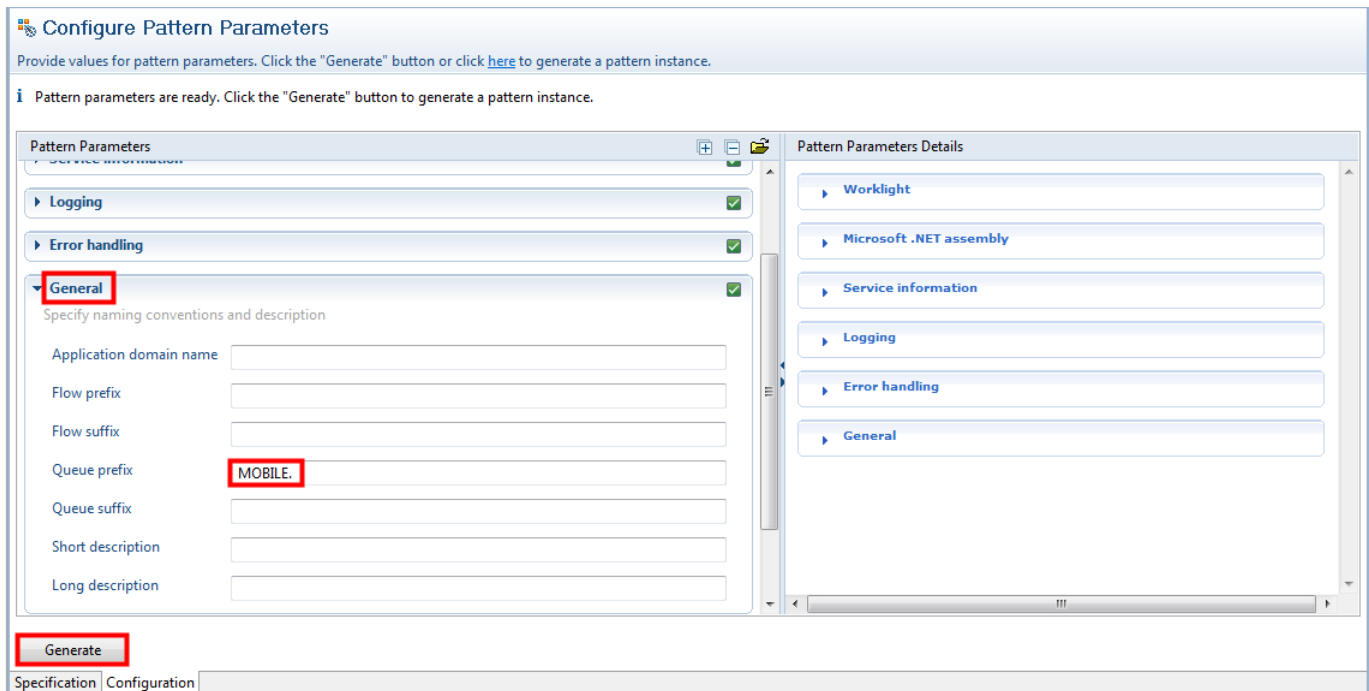
Pattern Parameters

- Worklight
- Microsoft .NET assembly
- Service information
- Logging**
 - Enable logging messages and specify their destination
 - Logging required * ☒
 - Log queue manager
 - Log queue * LOG
- Error handling
- General

Pattern Parameters Details

- Worklight
- Microsoft .NET assembly
- Service information
- Logging
- Error handling
- General

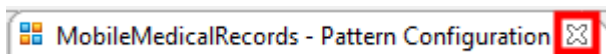
- __25. Collapse the **Logging** section.
- __26. Expand the **General** section.
- __27. Enter **MOBILE.** as the **Queue prefix**. **Please do not forget the period at the end of the prefix.**
- __28. Press the **Generate** button.



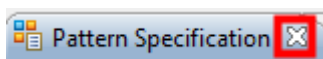
The pattern will take a couple of minutes to generate, after which the following projects will have been created in the toolkit navigator.

- Application: **MobileMedicalRecords_DotNetServiceApplicationFlows** – the main flow.
- Library: **MobileMedicalRecords_DotNetServiceLibraryFlows** – where you put your own flows – will not be overwritten if the pattern is regenerated.
- Independent Resource: **MobileMedicalRecords_TestApplication** – Worklight Adapters.

- __29. Close the **Pattern Configuration** editor.

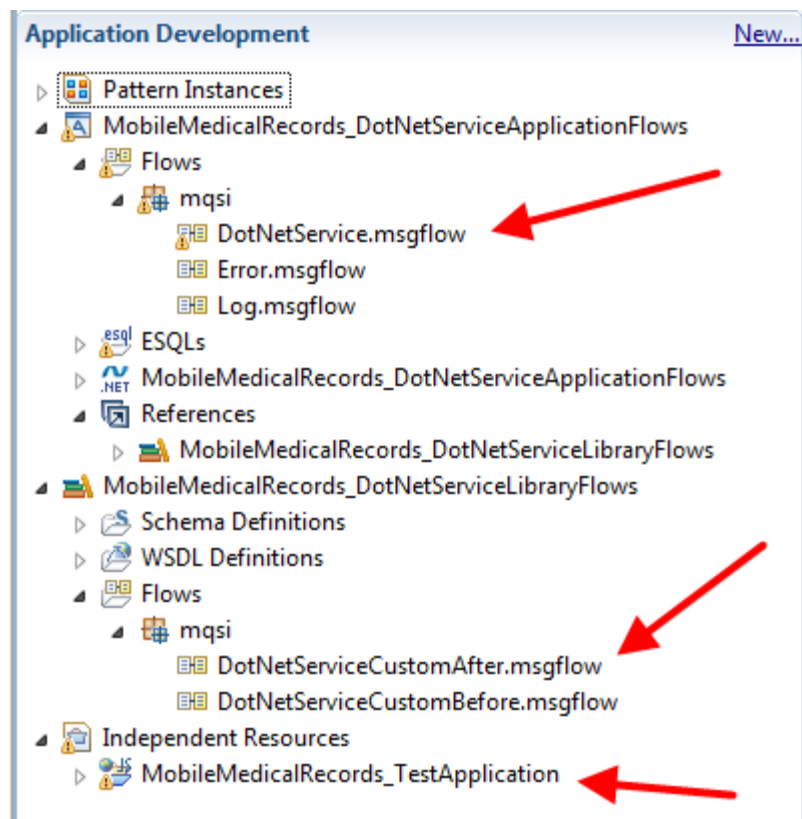


- __30. Close the **Pattern Specification** window.



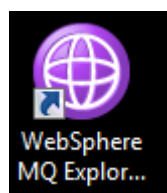
__31. Expand the generated projects. Observe the message flows that were generated.

A Worklight adapter project (**MobileMedicalRecords_TestApplication**) has been generated under **Independent Resources**.

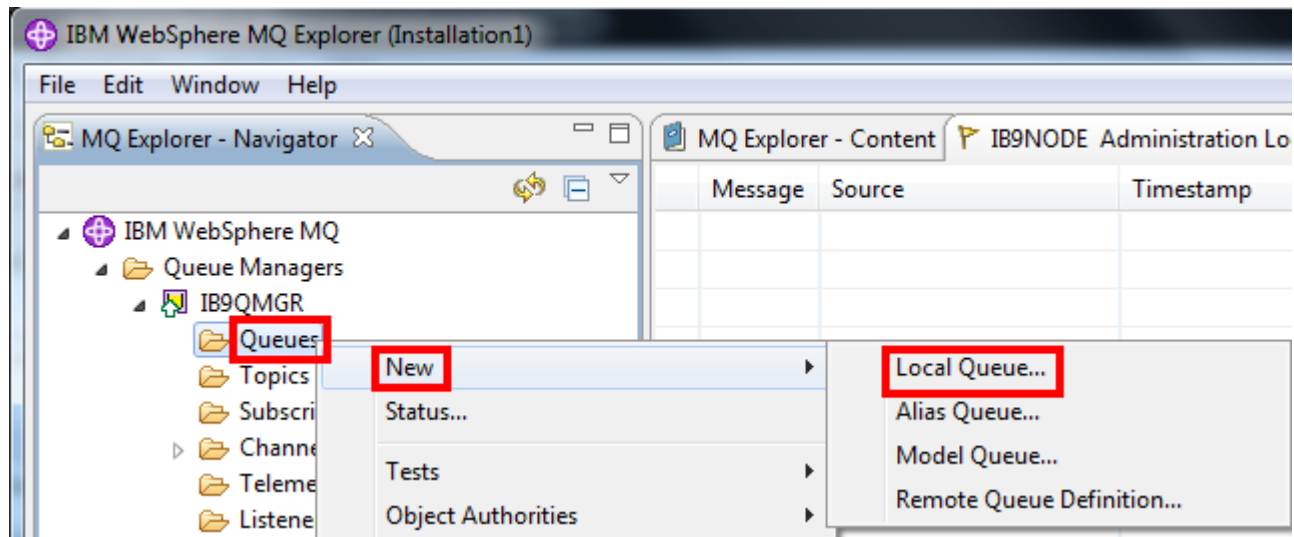


The message flows use MQ queues. The queues will be created next.

__32. Either return to or start MQ Explorer.

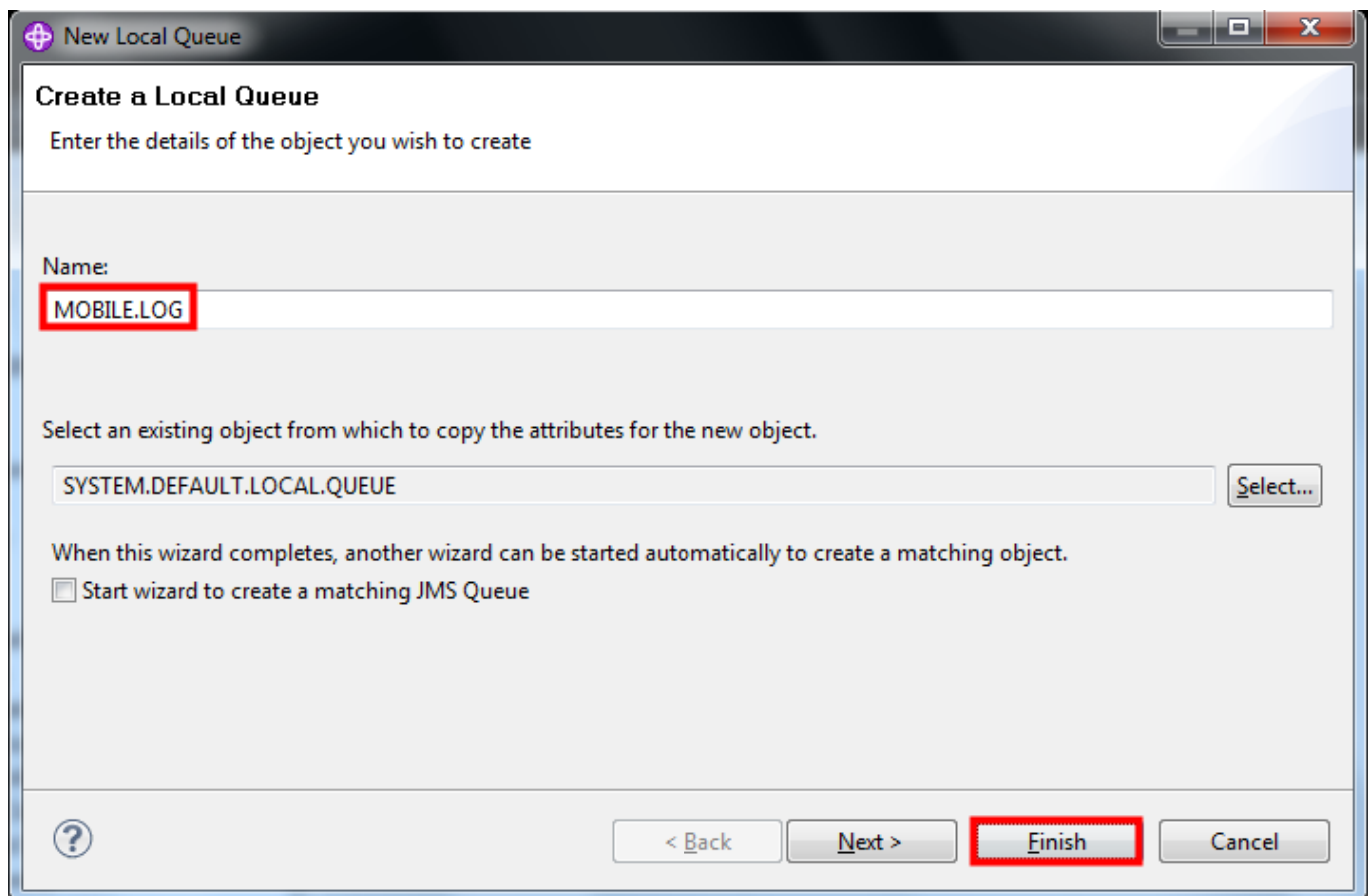


- __33. Expand **Queue Managers**→**IB9QMGR**.
- __34. Select the **Queues** folder.
- __35. Press the right mouse button.
- __36. Select **New**→**Local Queue** from the menu.

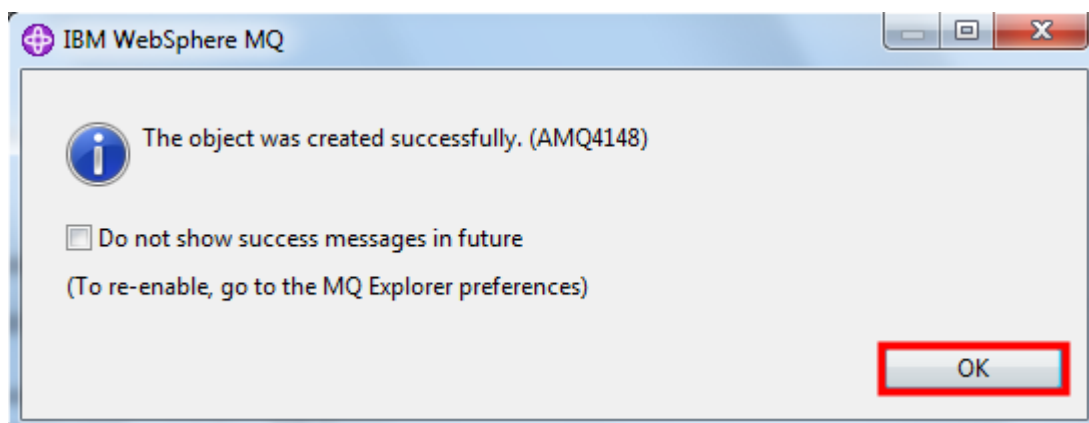


__37. Enter **MOBILE.LOG** as the **Name** of the queue.

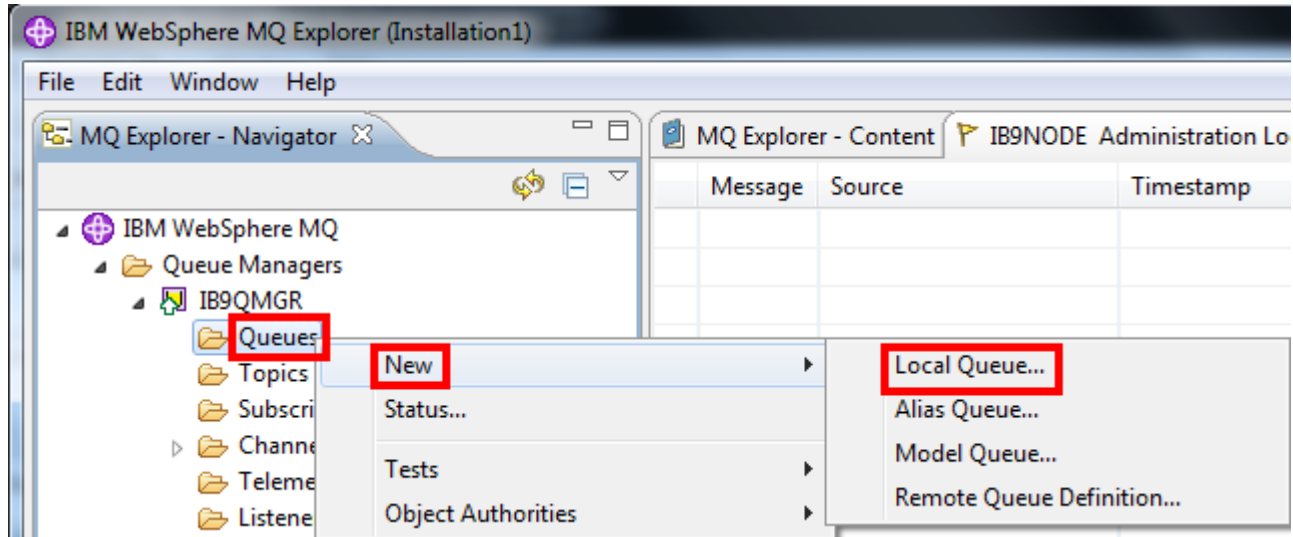
__38. Press the **Finish** button to create the queue.



__39. Press the **OK** button to dismiss the dialog box.

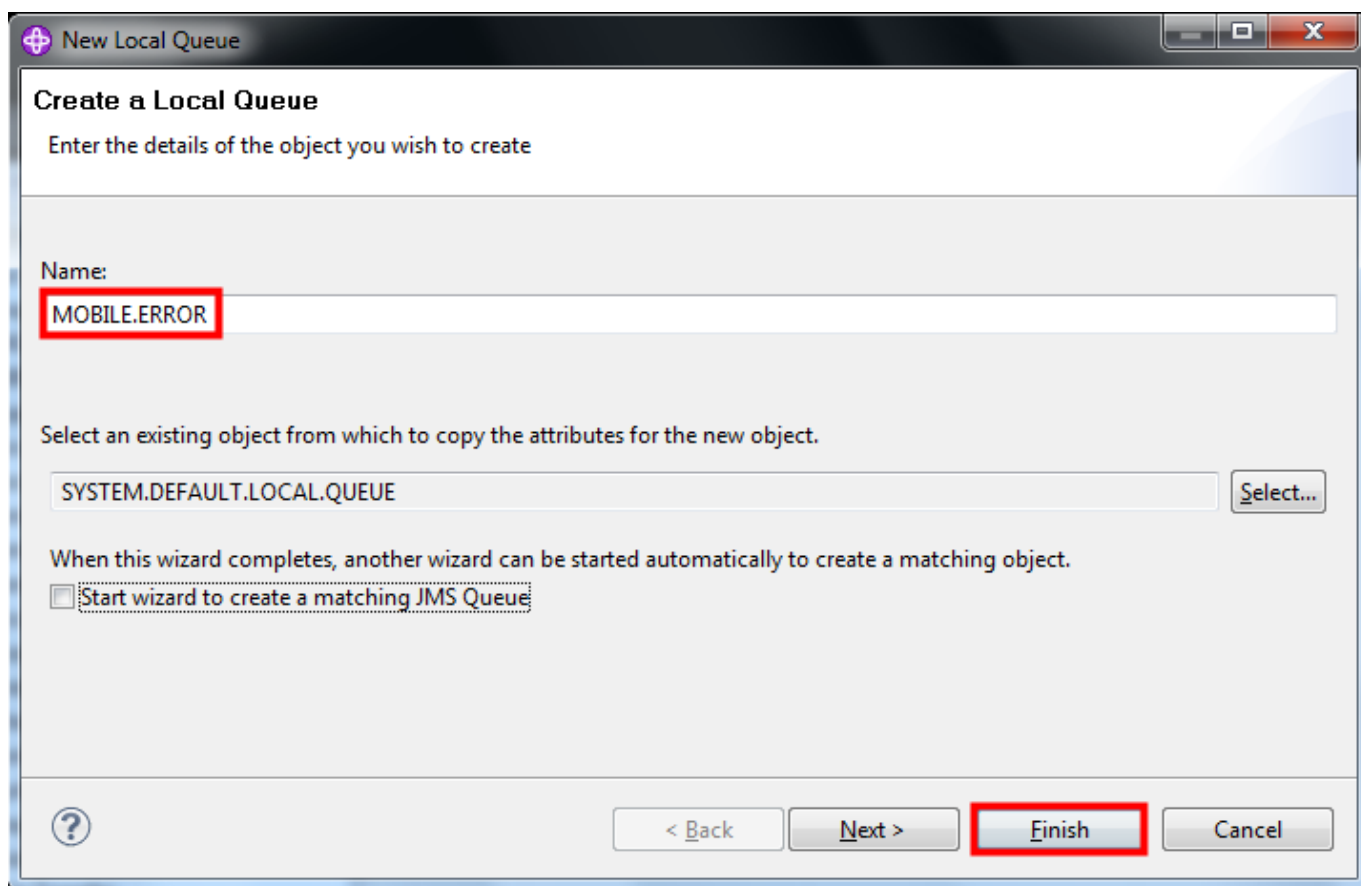


- __40. Select the **Queues** folder.
- __41. Press the right mouse button.
- __42. Select **New→Local Queue** from the menu.

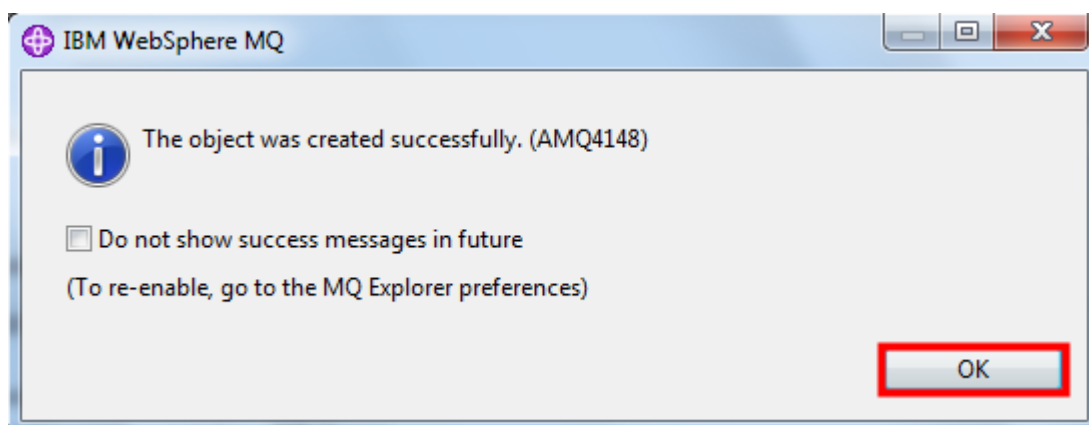


__43. Enter **MOBILE.ERROR** as the **Name** of the queue.

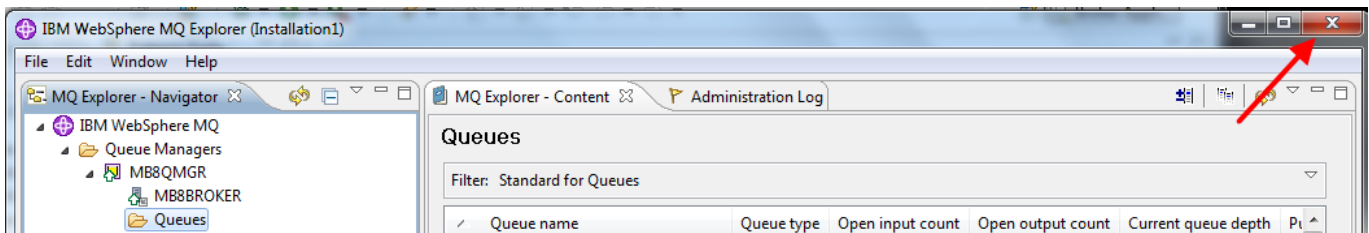
__44. Press the **Finish** button to create the queue.



__45. Press the **OK** button to dismiss the dialog box.



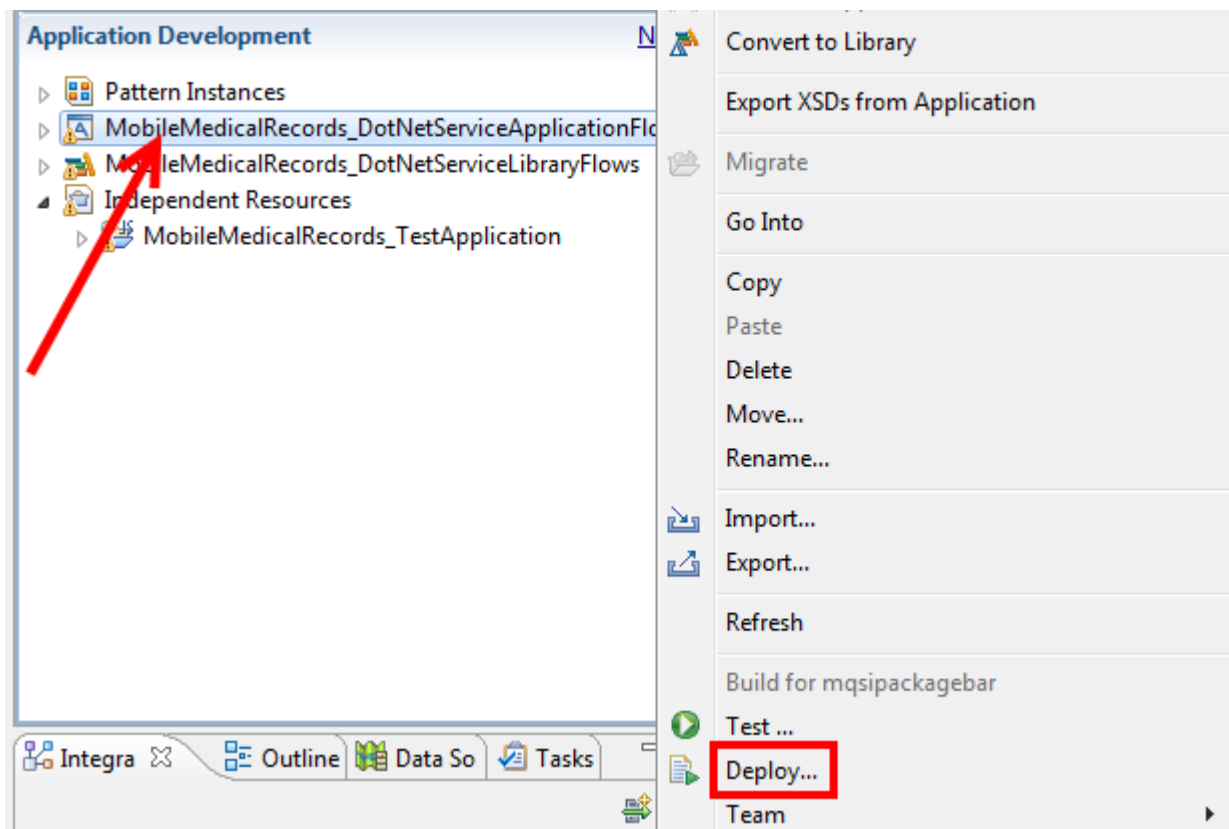
__46. Close the MQ Explorer.



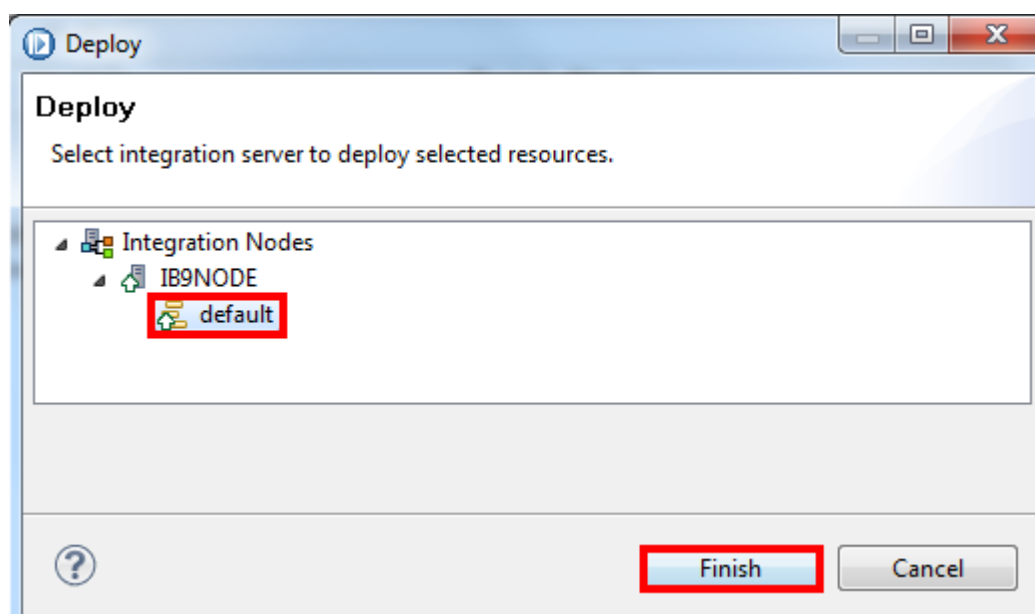
__47. Select the **MobileMedicalRecords_DotNetServiceApplicationFlows** application.

__48. Press the right mouse button.

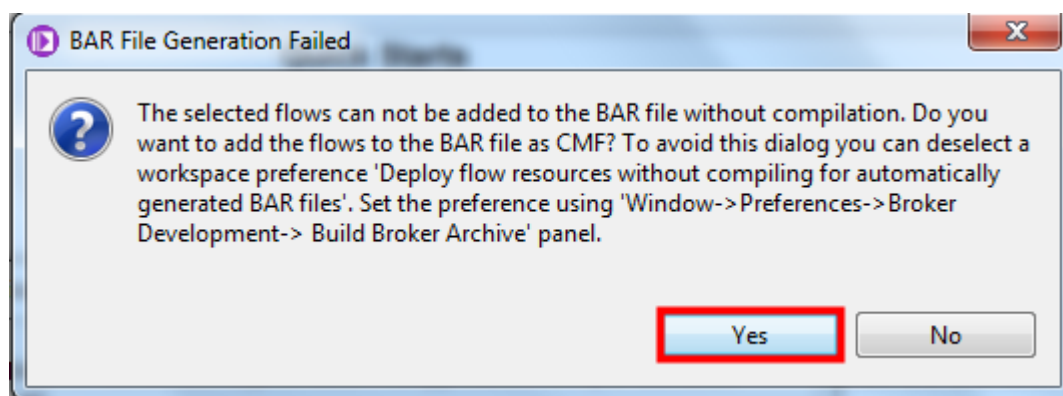
__49. Select **Deploy** from the menu.



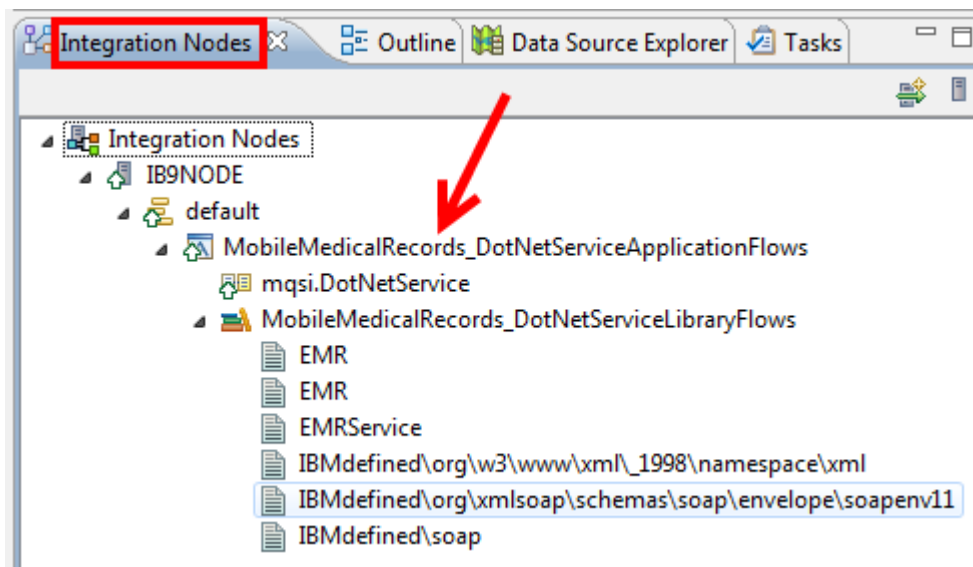
- __50. Select the **default** integration server.
- __51. Press the **Finish** button to initiate the deployment.



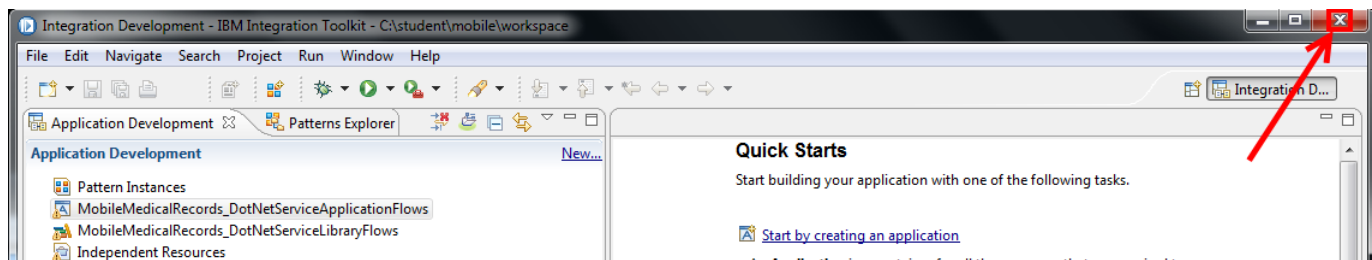
- __52. Press the **Yes** button to dismiss the warning dialog box.



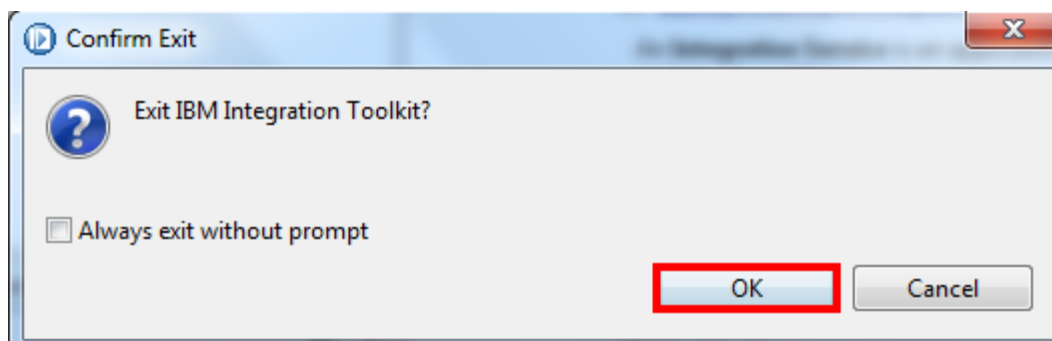
The deployed application should be visible in the **Integration Nodes** view.



__53. Close the Integration toolkit.



__54. Press the **OK** button to confirm the exit.

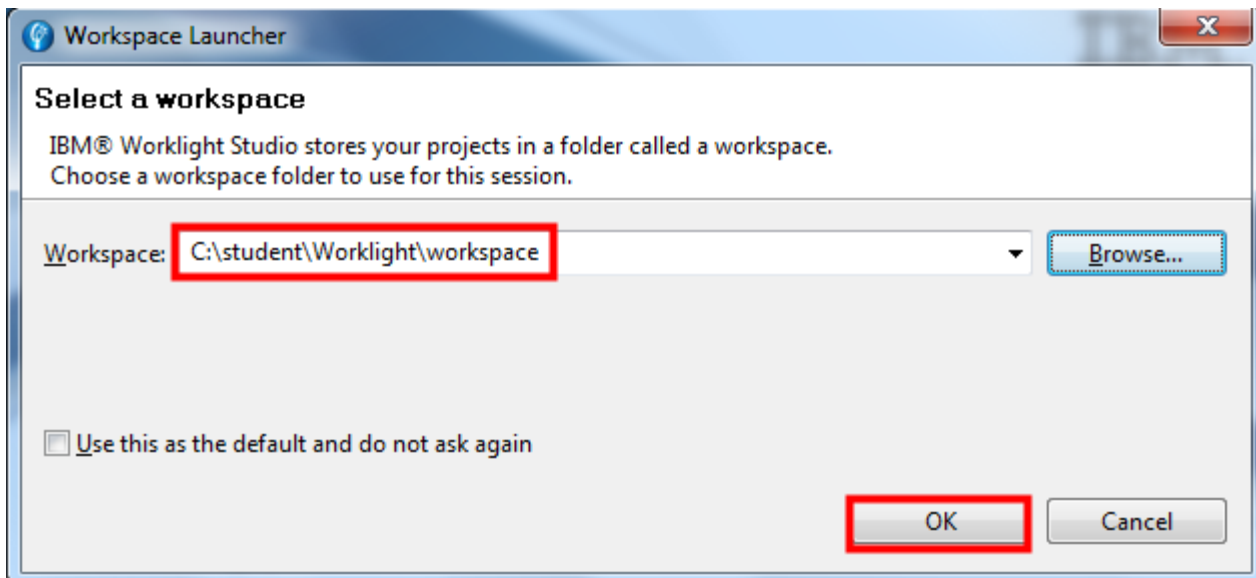


6.3 Deploy the Worklight adapters and test the application

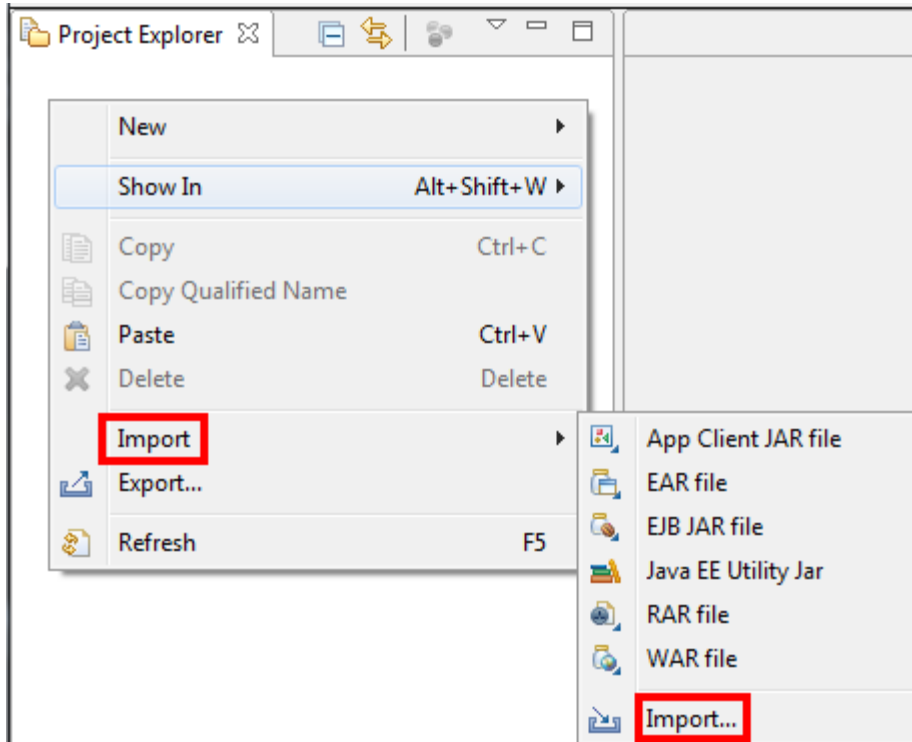
1. Open the IBM Worklight Studio, using the icon on the main window.



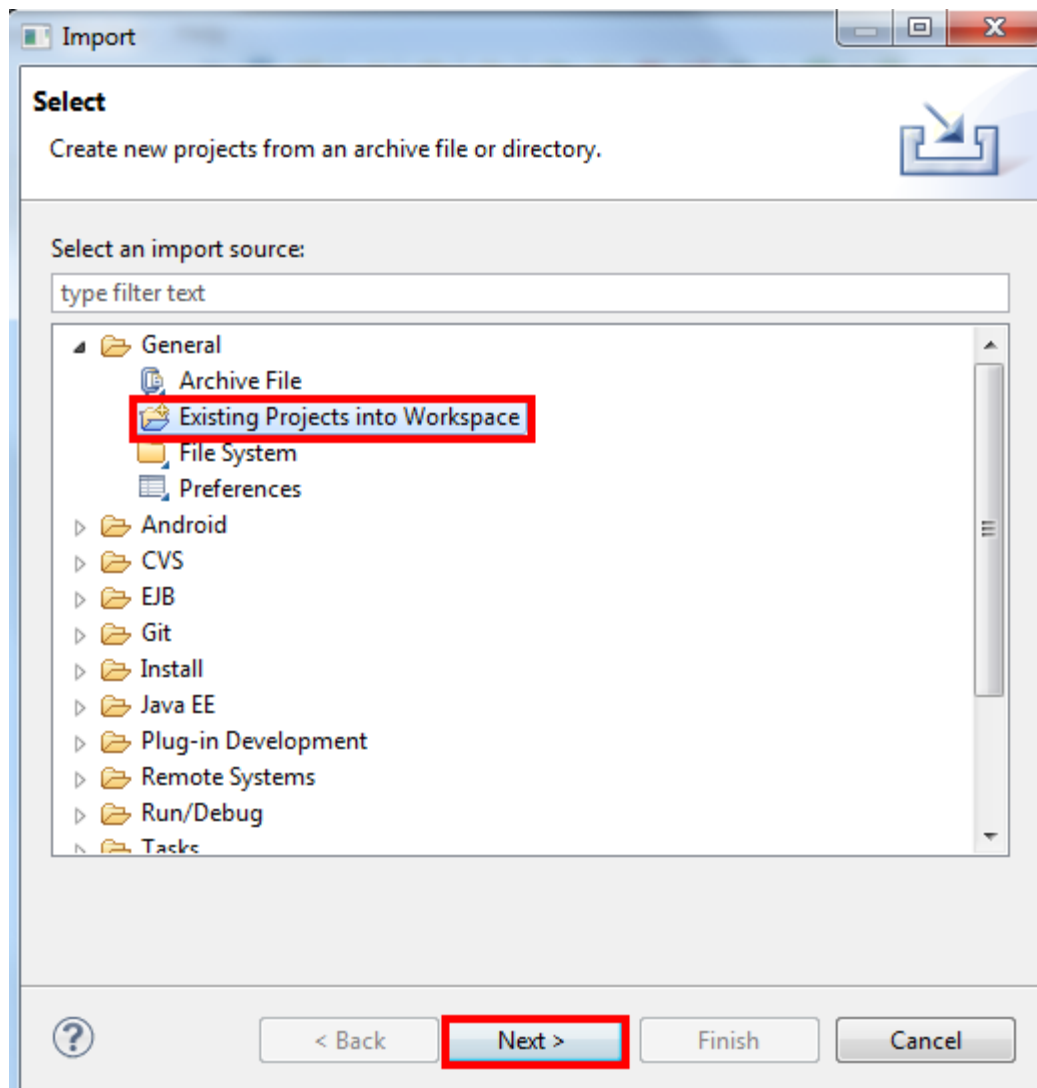
2. Select the **C:\student\Worklight\workspace** directory.
3. Press the **OK** button to open the workbench.



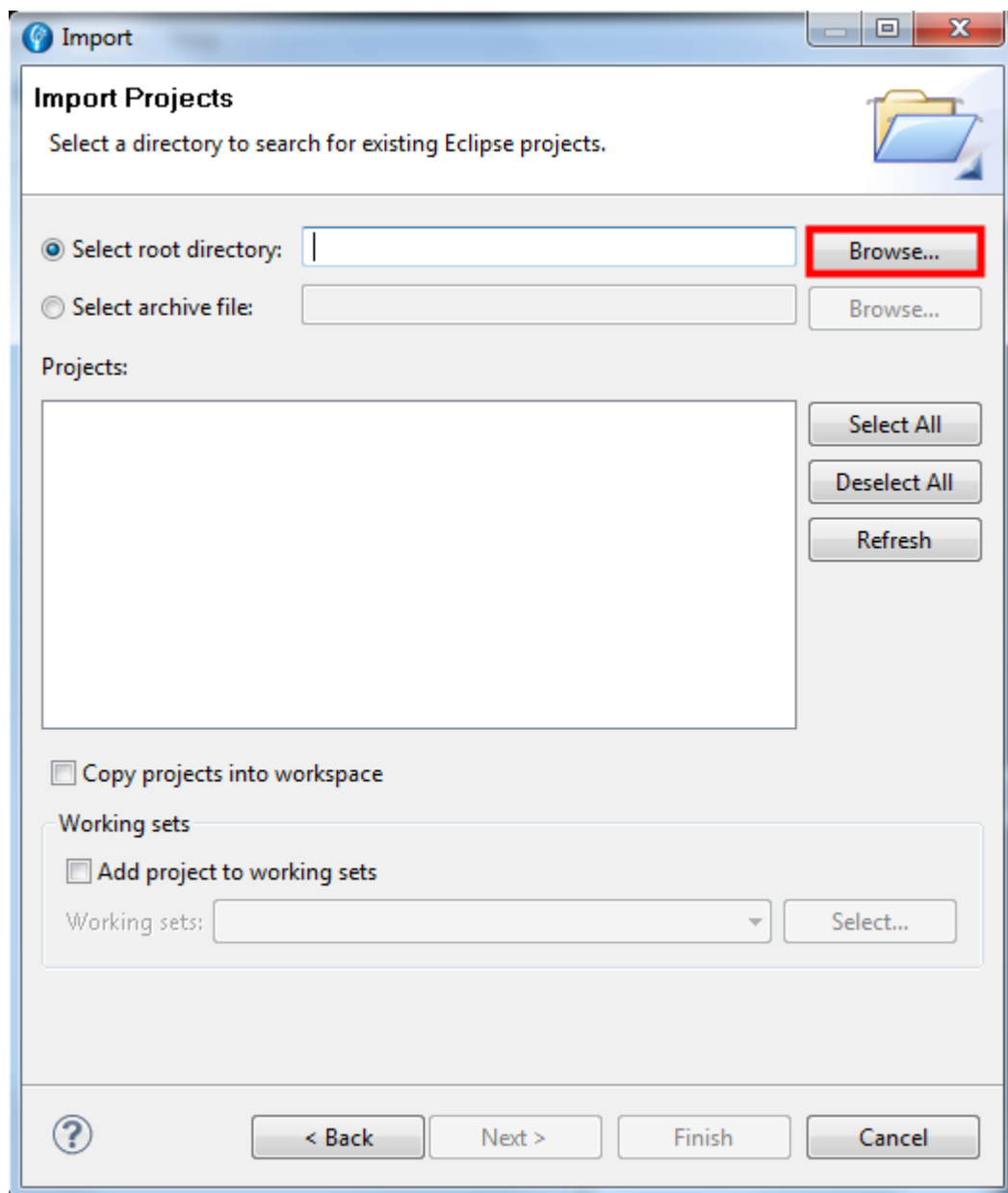
- __4. In the **Project Explorer**, point to a blank area.
- __5. Press the right mouse button.
- __6. Select **Import**→**Import...** from the menu.



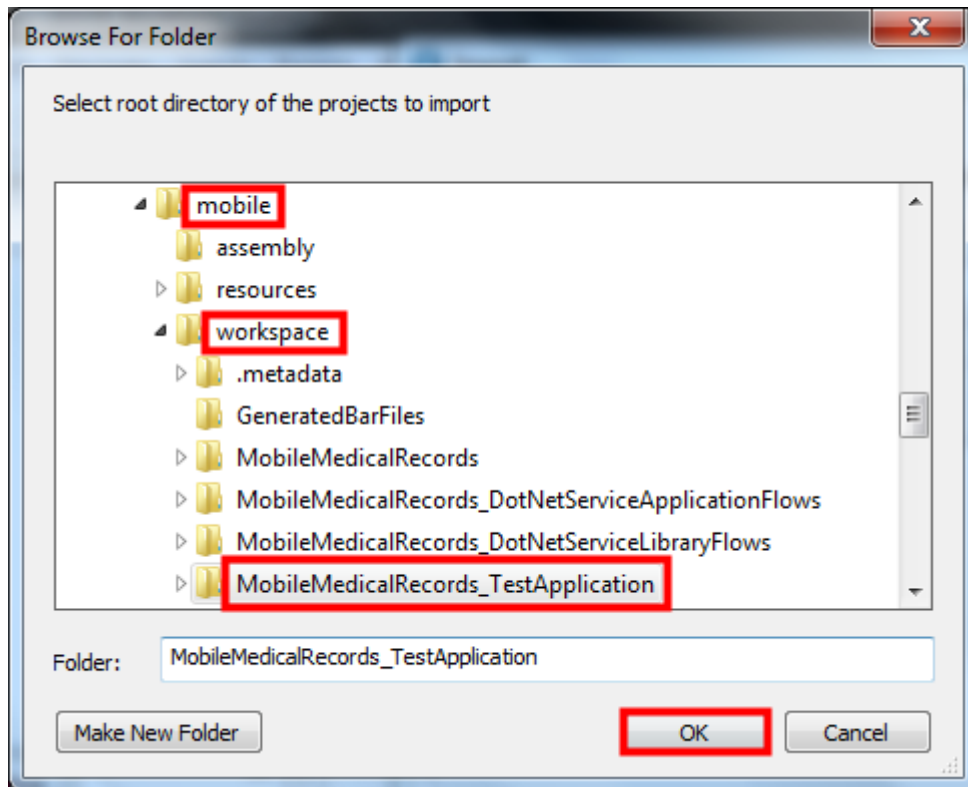
- __7. Expand the **General** folder.
- __8. Select **Existing Projects into Workspace**.
- __9. Press the **Next** button.



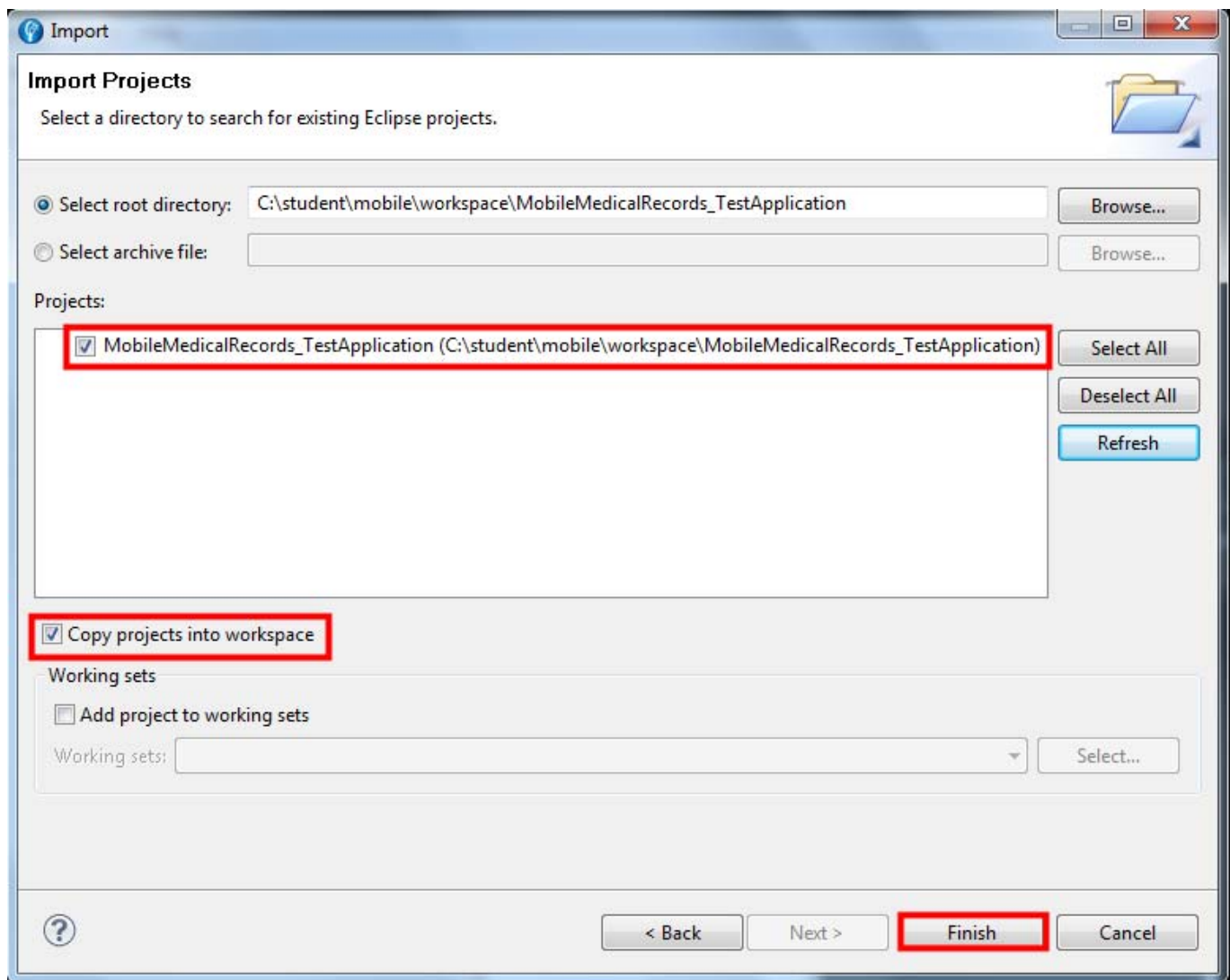
___10. Press the **Browse** button next to the **Select root directory** text box.



- __11. Navigate to the **C:\student\mobile\workspace\MobileMedicalRecords_TestApplication** project.
- __12. Press the **OK** button.



- __13. Select the **Copy projects into workspace** check box.
- __14. Press the **Finish** button to start the import.

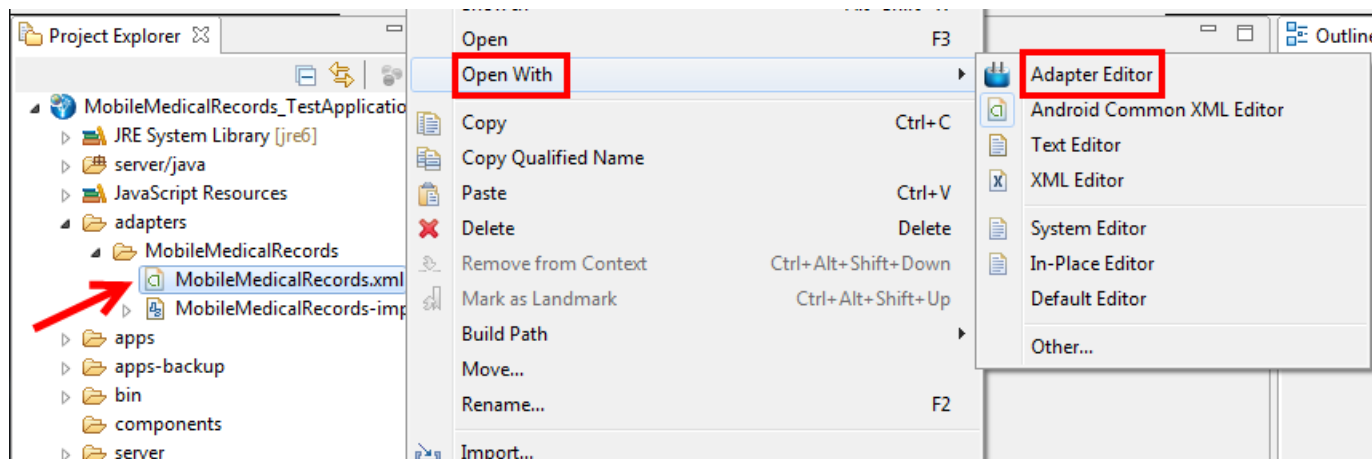


__15. Expand **MobileMedicalRecords_TestApplication**→**adapters**→**MobileMedicalRecords**.

__16. Select the **MobileMedicalRecords.xml** file.

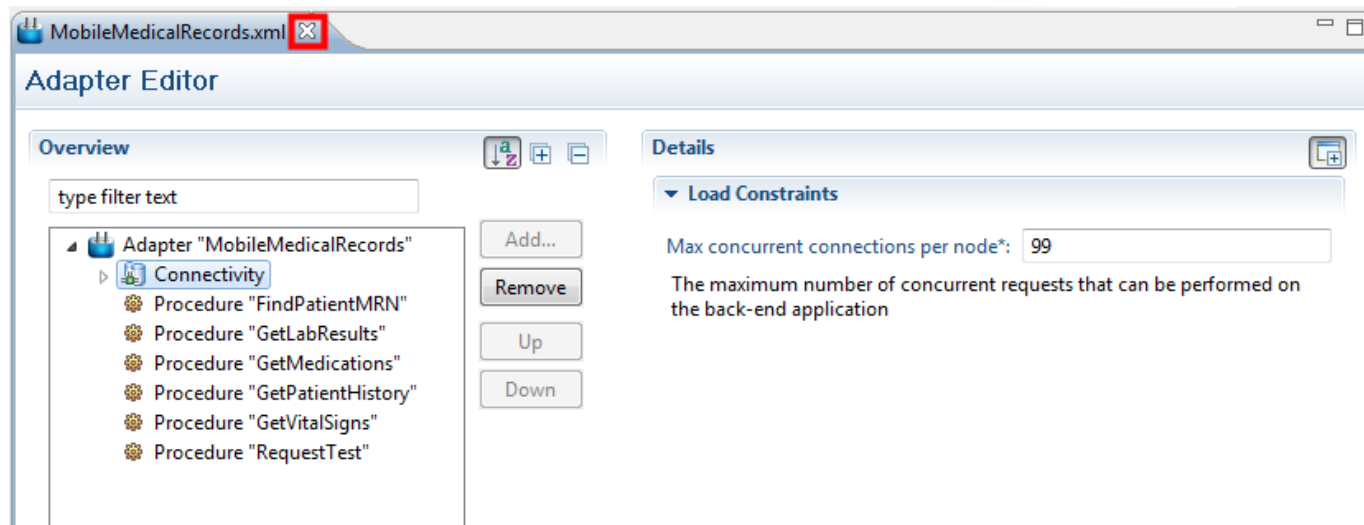
__17. Press the right mouse button.

__18. Select **Open With**→**Adapter Editor** from the menu.

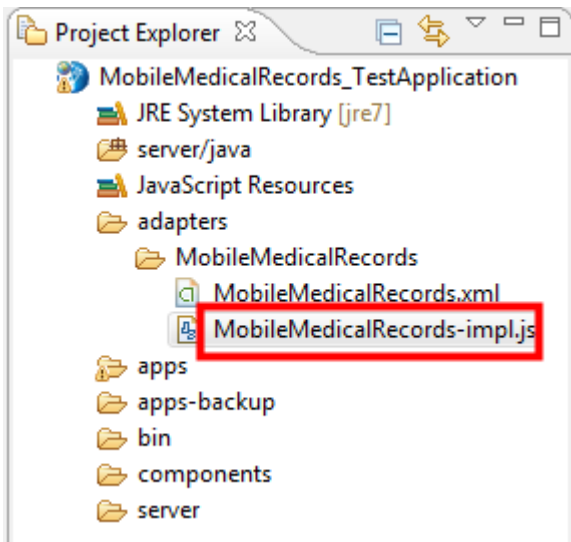


__19. Examine the contents of the adapter XML file.

__20. When finished close the editor.

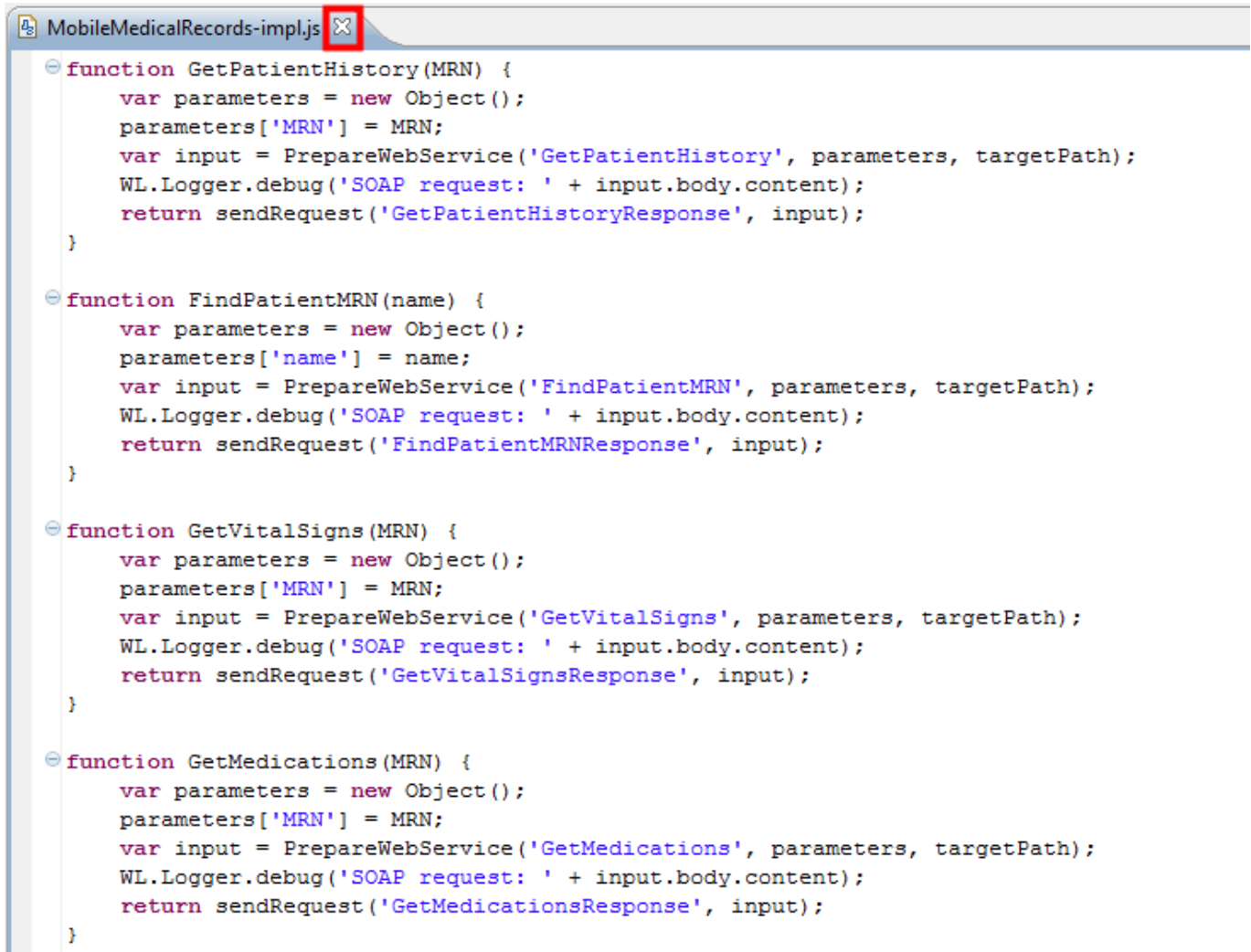


__21. Double-click the **MobileMedicalRecords-impl.js** java script file.



__22. Scroll down so that some of the individual methods are visible.

__23. After examining the Java script file close the editor pane.



```
MobileMedicalRecords-impl.js
function GetPatientHistory(MRN) {
    var parameters = new Object();
    parameters['MRN'] = MRN;
    var input = PrepareWebService('GetPatientHistory', parameters, targetPath);
    WL.Logger.debug('SOAP request: ' + input.body.content);
    return sendRequest('GetPatientHistoryResponse', input);
}

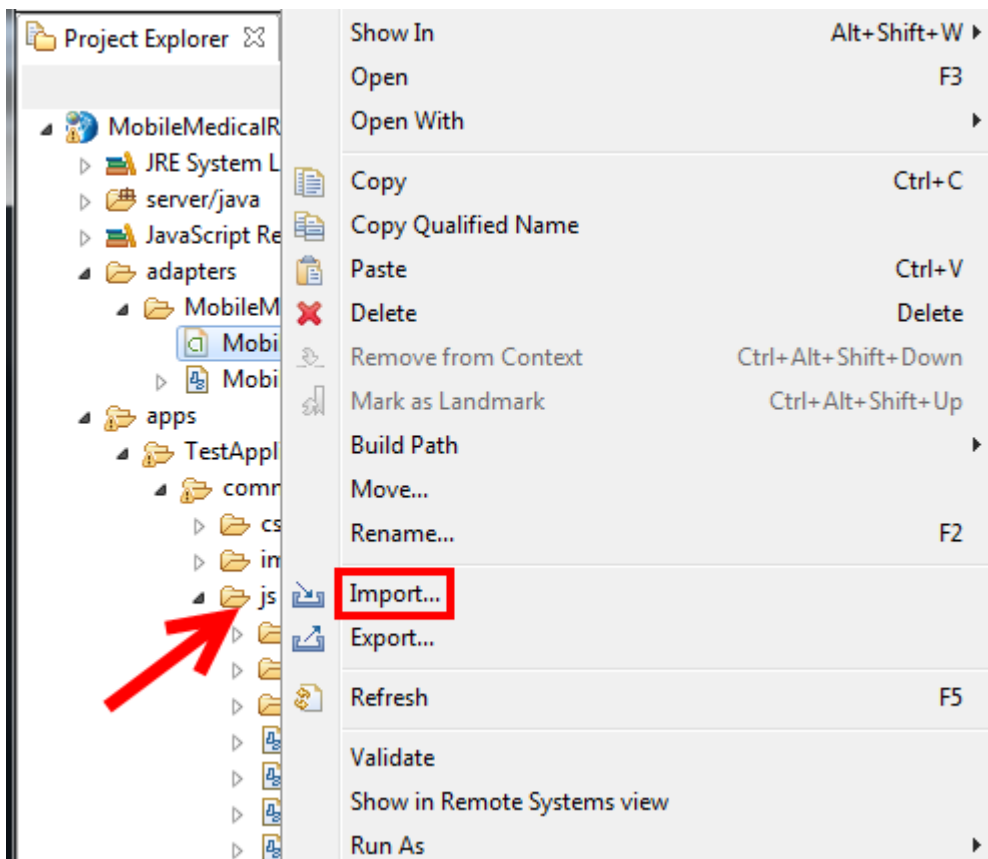
function FindPatientMRN(name) {
    var parameters = new Object();
    parameters['name'] = name;
    var input = PrepareWebService('FindPatientMRN', parameters, targetPath);
    WL.Logger.debug('SOAP request: ' + input.body.content);
    return sendRequest('FindPatientMRNResponse', input);
}

function GetVitalSigns(MRN) {
    var parameters = new Object();
    parameters['MRN'] = MRN;
    var input = PrepareWebService('GetVitalSigns', parameters, targetPath);
    WL.Logger.debug('SOAP request: ' + input.body.content);
    return sendRequest('GetVitalSignsResponse', input);
}

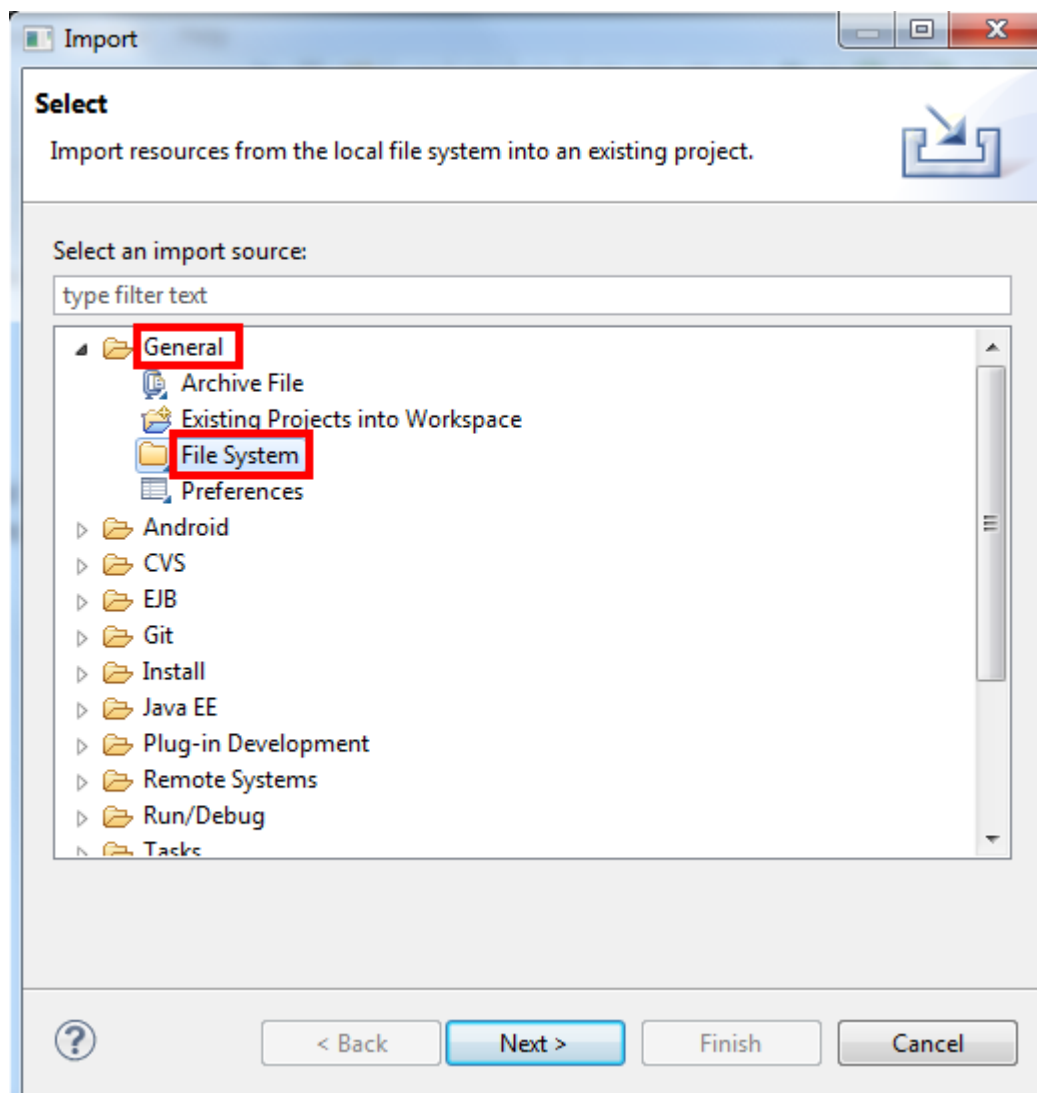
function GetMedications(MRN) {
    var parameters = new Object();
    parameters['MRN'] = MRN;
    var input = PrepareWebService('GetMedications', parameters, targetPath);
    WL.Logger.debug('SOAP request: ' + input.body.content);
    return sendRequest('GetMedicationsResponse', input);
}
```


The TestApplication application was created with an earlier version of Worklight. There have been subsequent changes in Worklight that must now be corrected so that the application can be used with a recent Worklight studio.

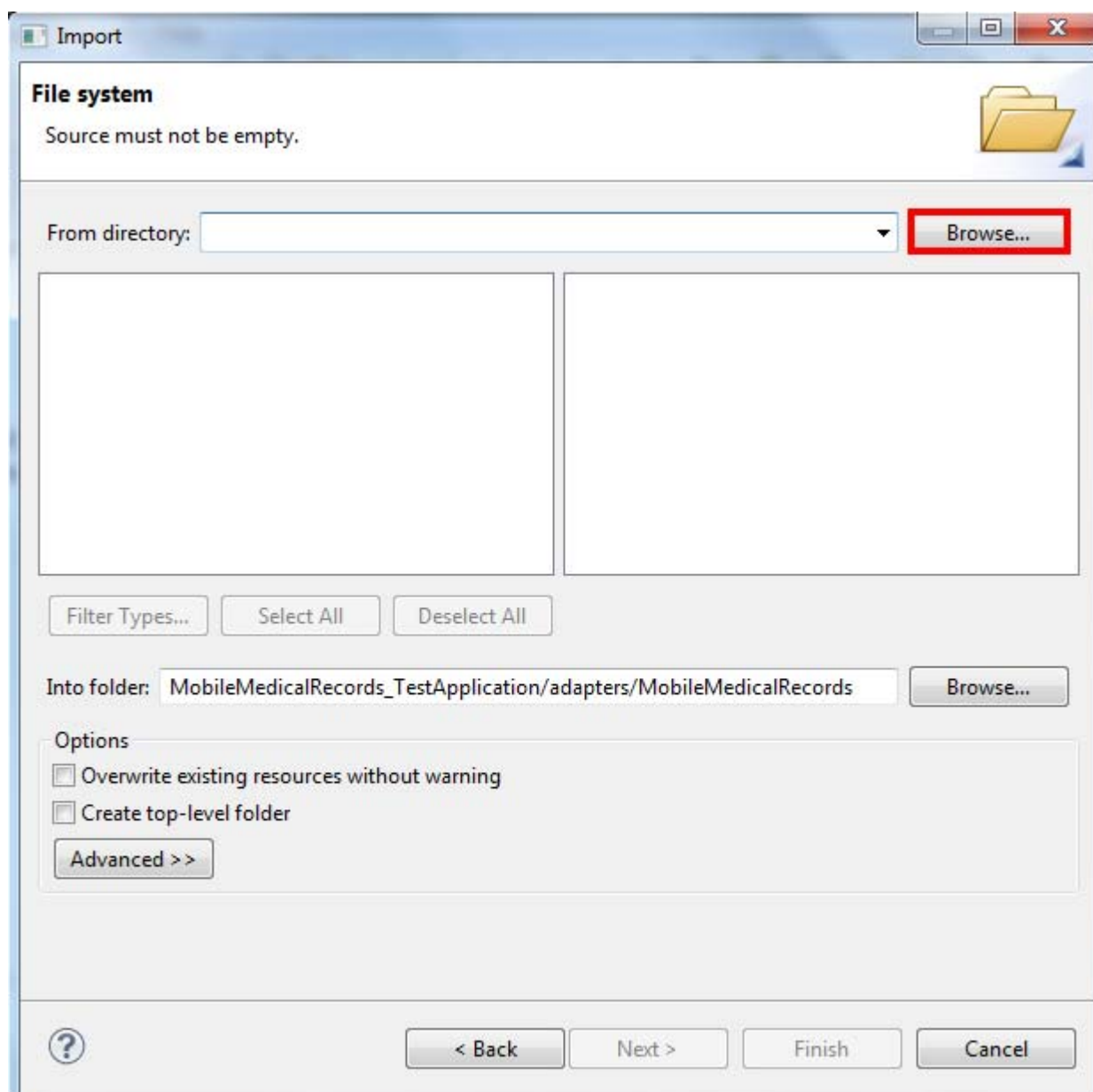
- __24. Expand **MobileMedicalRecords_TestApplication→apps→TestApplication→Common→js**.
- __25. Select the **js** folder.
- __26. Press the right mouse button.
- __27. Select **Import** from the menu.



- __28. Expand the **General** folder.
- __29. Select **File System**.
- __30. Press the **Next** button.



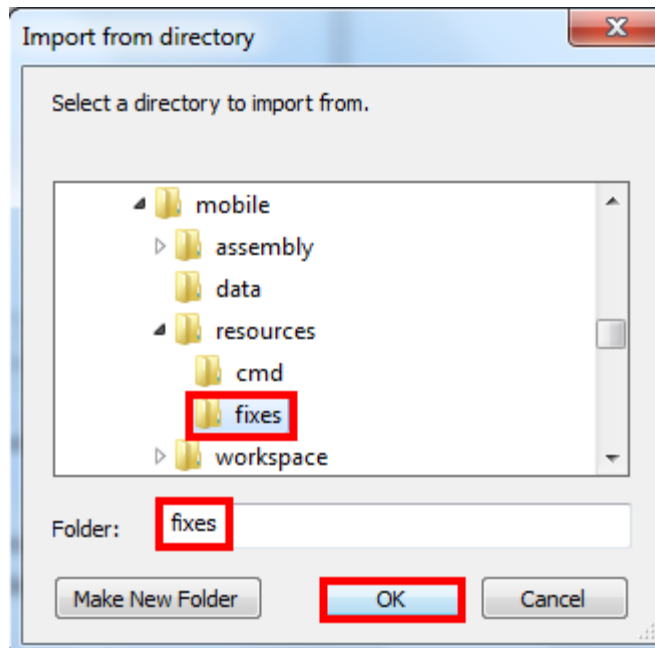
__31. Press the **Browse** button to the right of **From directory**.



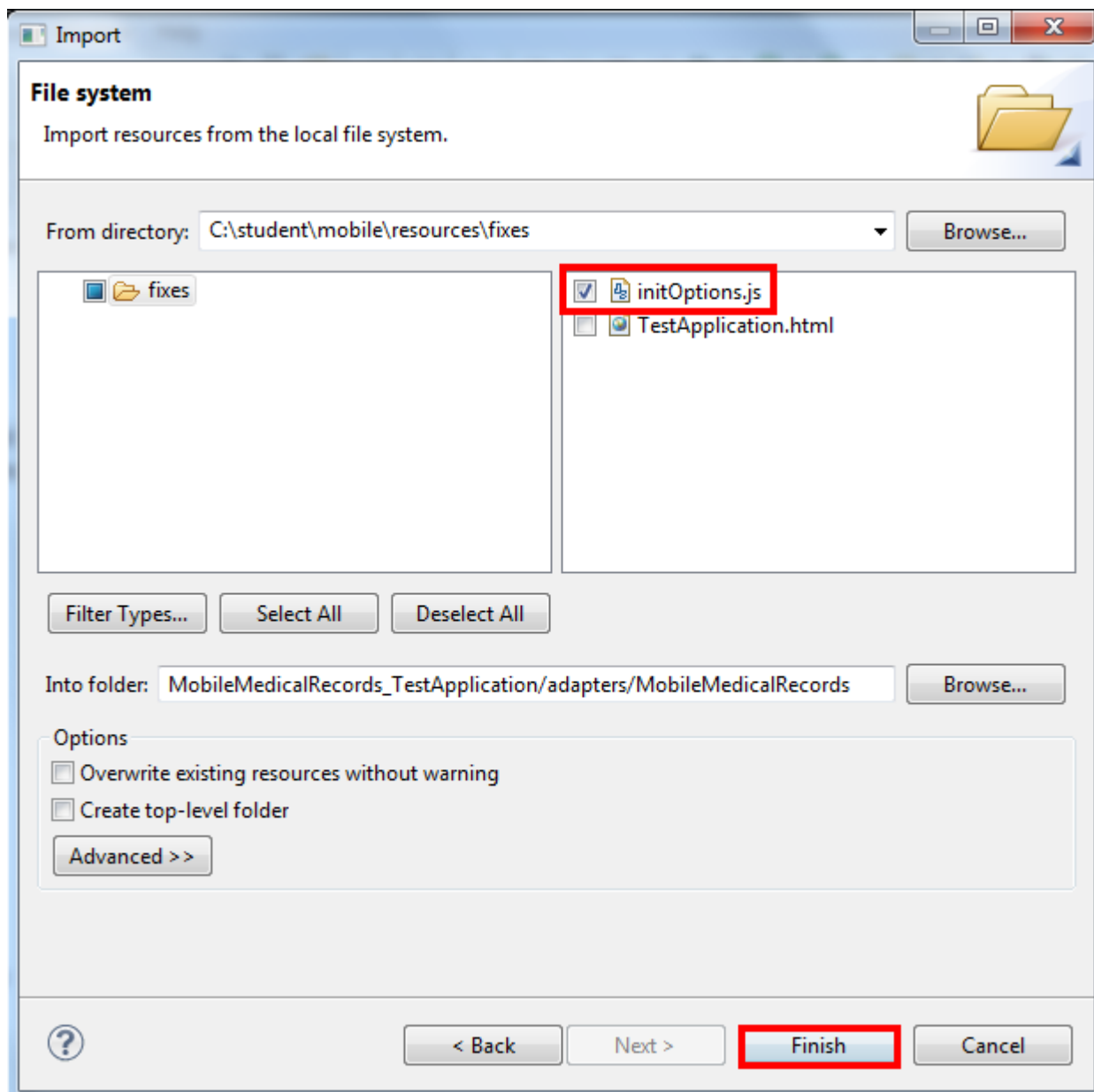
__32. Navigate to the **C:\student\mobile\resources** directory.

__33. Select the **fixes** directory.

__34. Press the **OK** button.

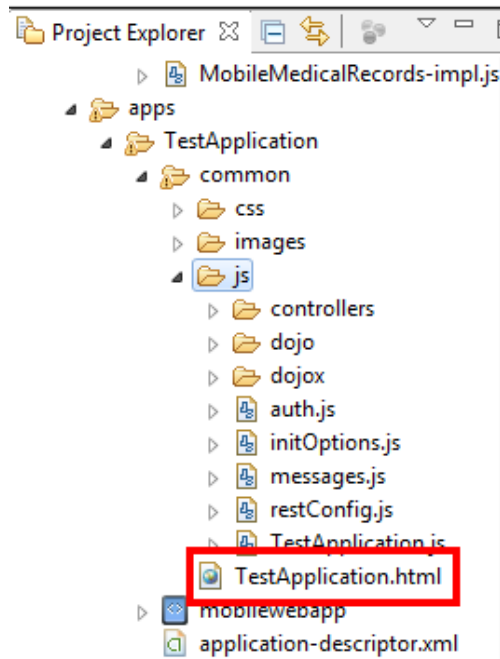


- __35. Select the **initOptions.js** check box.
- __36. Press the **Finish** button to perform the import.



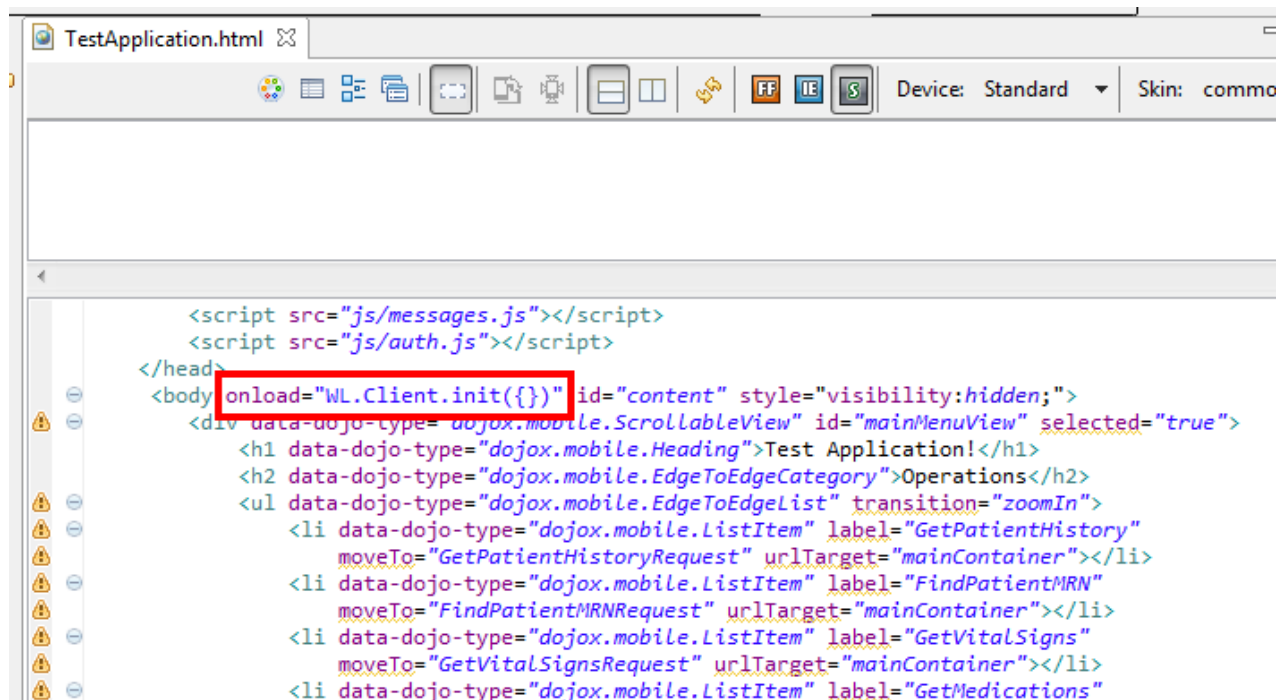
The TestApplication.html file must also be fixed. A complete replacement file is available in the **C:\student\mobile\resources\fixes** directory.

__37. In the Project Explorer double-click the **TestApplication.html** file.

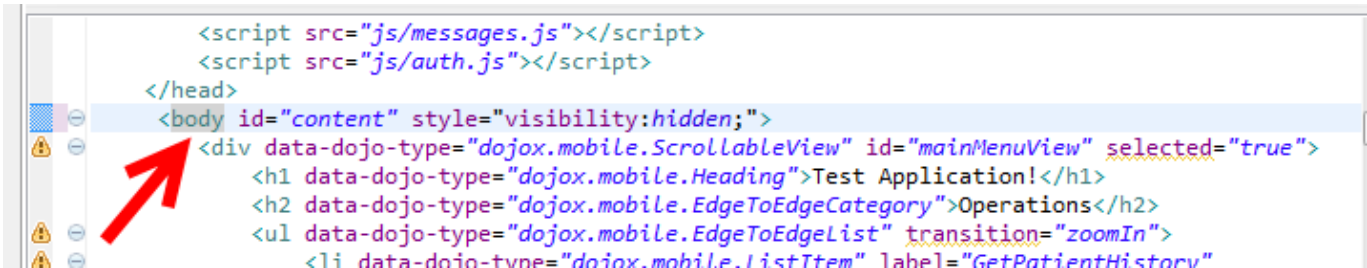


__38. Scroll down so the **Body** element is visible.

__39. Delete the highlighted text (**onload="WL.Client.init({})"**).

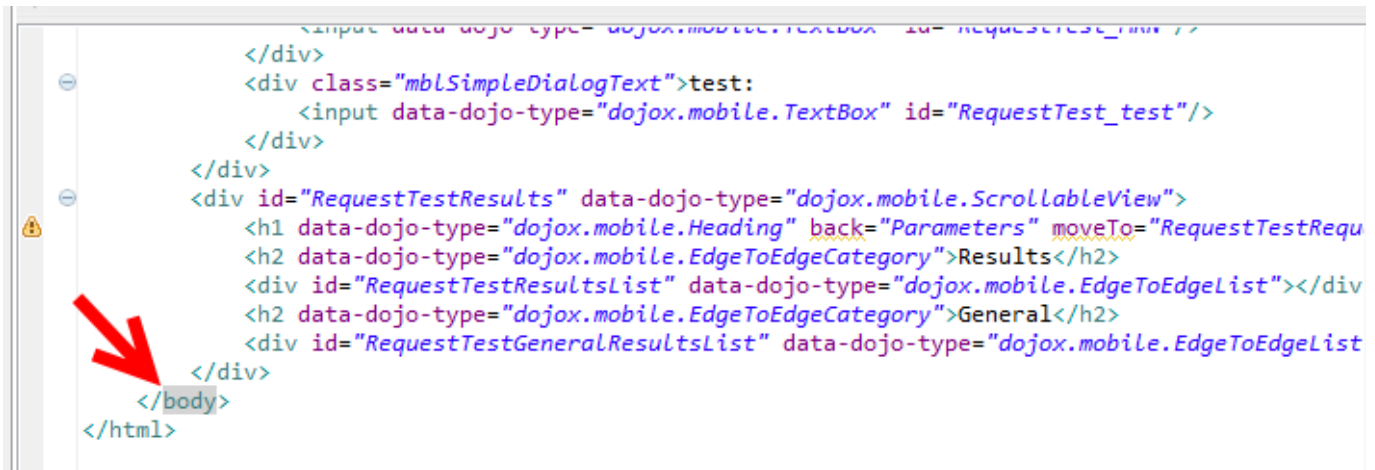


__40. The result should look like the following.



```
<script src="js/messages.js"></script>
<script src="js/auth.js"></script>
</head>
<body id="content" style="visibility:hidden;">
  <div data-dojo-type="dojox.mobile.ScrollableView" id="mainMenuView" selected="true">
    <h1 data-dojo-type="dojox.mobile.Heading">Test Application!</h1>
    <h2 data-dojo-type="dojox.mobile.EdgeToEdgeCategory">Operations</h2>
    <ul data-dojo-type="dojox.mobile.EdgeToEdgeList" transition="zoomIn">
      <li data-dojo-type="dojox.mobile.ListItem" label="GetPatientHistory">
```

__41. Scroll down to the end of the body.



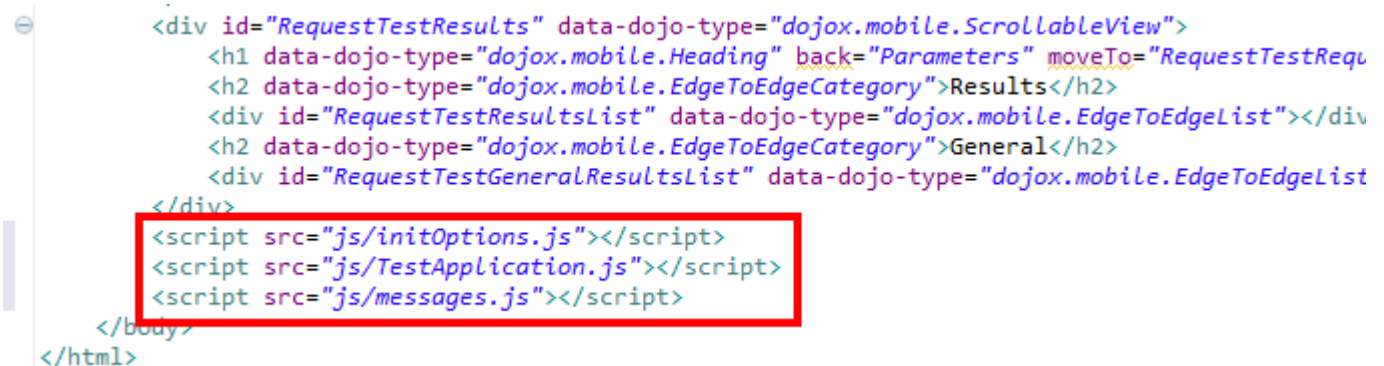
```
<input data-dojo-type="dojox.mobile.TextBox" id="RequestTest_test" />
</div>
<div class="mblSimpleDialogText">test:
  <input data-dojo-type="dojox.mobile.TextBox" id="RequestTest_test" />
</div>
</div>
<div id="RequestTestResults" data-dojo-type="dojox.mobile.ScrollableView">
  <h1 data-dojo-type="dojox.mobile.Heading" back="Parameters" moveTo="RequestTestRequ
  <h2 data-dojo-type="dojox.mobile.EdgeToEdgeCategory">Results</h2>
  <div id="RequestTestResultsList" data-dojo-type="dojox.mobile.EdgeToEdgeList"></div>
  <h2 data-dojo-type="dojox.mobile.EdgeToEdgeCategory">General</h2>
  <div id="RequestTestGeneralResultsList" data-dojo-type="dojox.mobile.EdgeToEdgeList
</div>
</body>
</html>
```

__42. Insert the following statements before the **</body>** statement.

```
<script src="js/initOptions.js"></script>
```

```
<script src="js/TestApplication.js">
```

```
<script src="js/messages.js"></script>
```

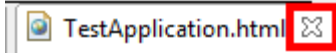


```
<div id="RequestTestResults" data-dojo-type="dojox.mobile.ScrollableView">
  <h1 data-dojo-type="dojox.mobile.Heading" back="Parameters" moveTo="RequestTestRequ
  <h2 data-dojo-type="dojox.mobile.EdgeToEdgeCategory">Results</h2>
  <div id="RequestTestResultsList" data-dojo-type="dojox.mobile.EdgeToEdgeList"></div>
  <h2 data-dojo-type="dojox.mobile.EdgeToEdgeCategory">General</h2>
  <div id="RequestTestGeneralResultsList" data-dojo-type="dojox.mobile.EdgeToEdgeList
</div>
<script src="js/initOptions.js"></script>
<script src="js/TestApplication.js"></script>
<script src="js/messages.js"></script>
</body>
</html>
```



__43. Save the modified html file (**Ctrl+S**).

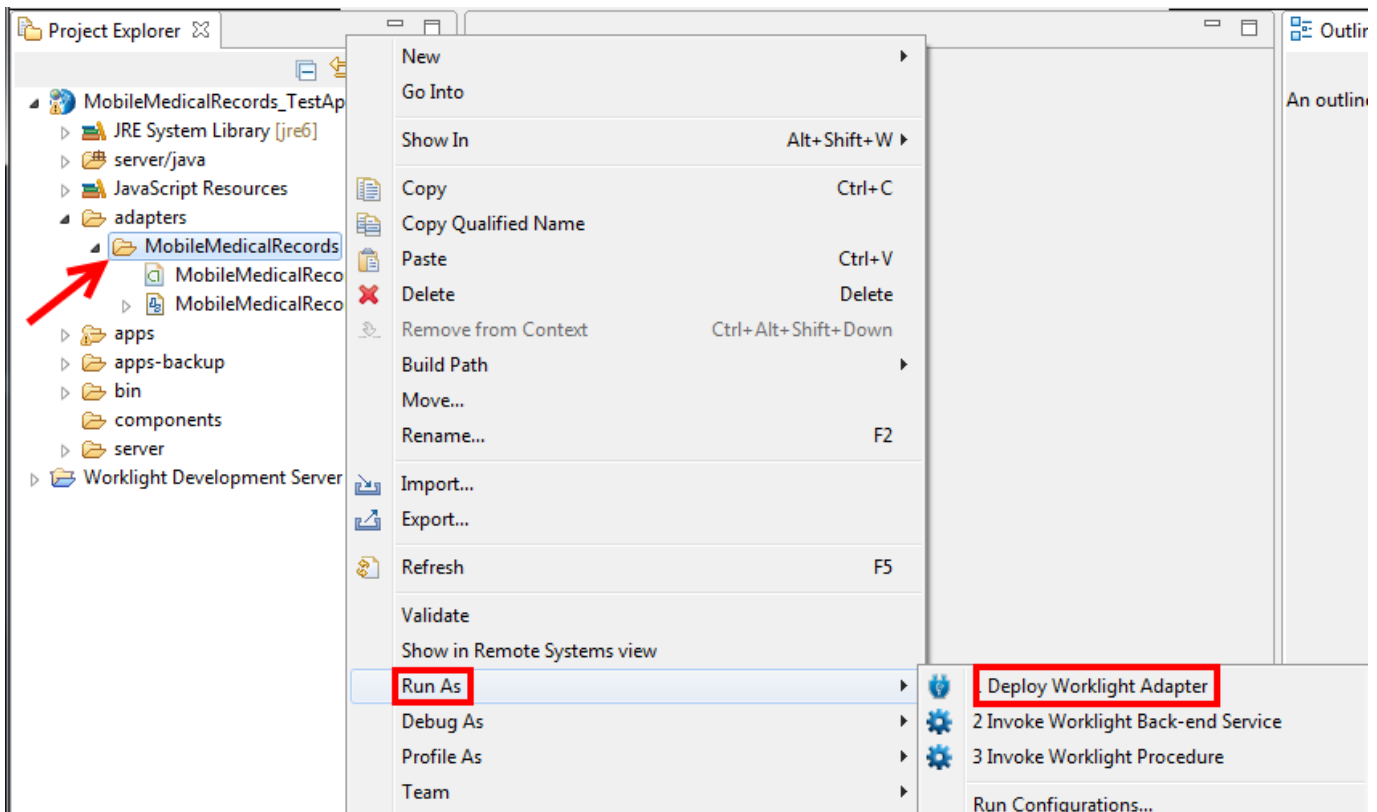
__44. Close the editor.



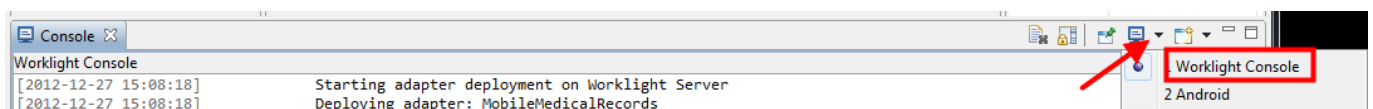
__45. Select the **MobileMedicalRecords** adapter.

__46. Press the right mouse button.

__47. Select **Run As→Deploy Worklight Adapter** from the menu.

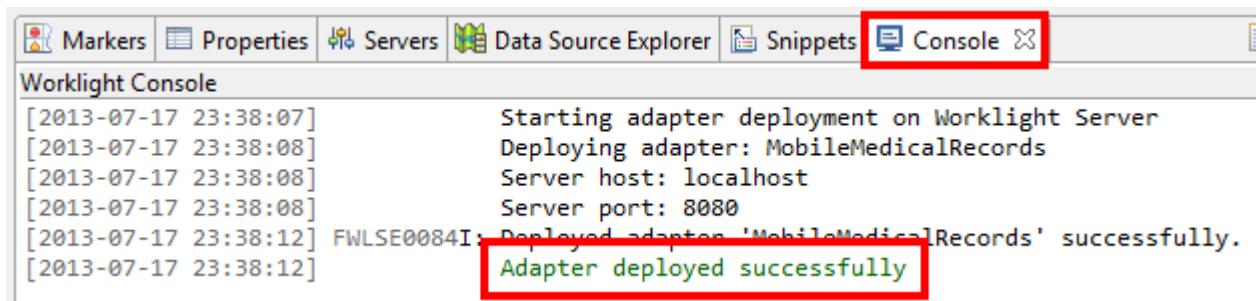


__48. Use the console icon to select the **Worklight Console** (if necessary).



The results of the deployment should be visible in the console.

__49. Confirm that the adapter deployment was successful.



```
Worklight Console
[2013-07-17 23:38:07] Starting adapter deployment on Worklight Server
[2013-07-17 23:38:08] Deploying adapter: MobileMedicalRecords
[2013-07-17 23:38:08] Server host: localhost
[2013-07-17 23:38:08] Server port: 8080
[2013-07-17 23:38:12] FWLSE0084I: Deployed adapter 'MobileMedicalRecords' successfully.
[2013-07-17 23:38:12] Adapter deployed successfully
```

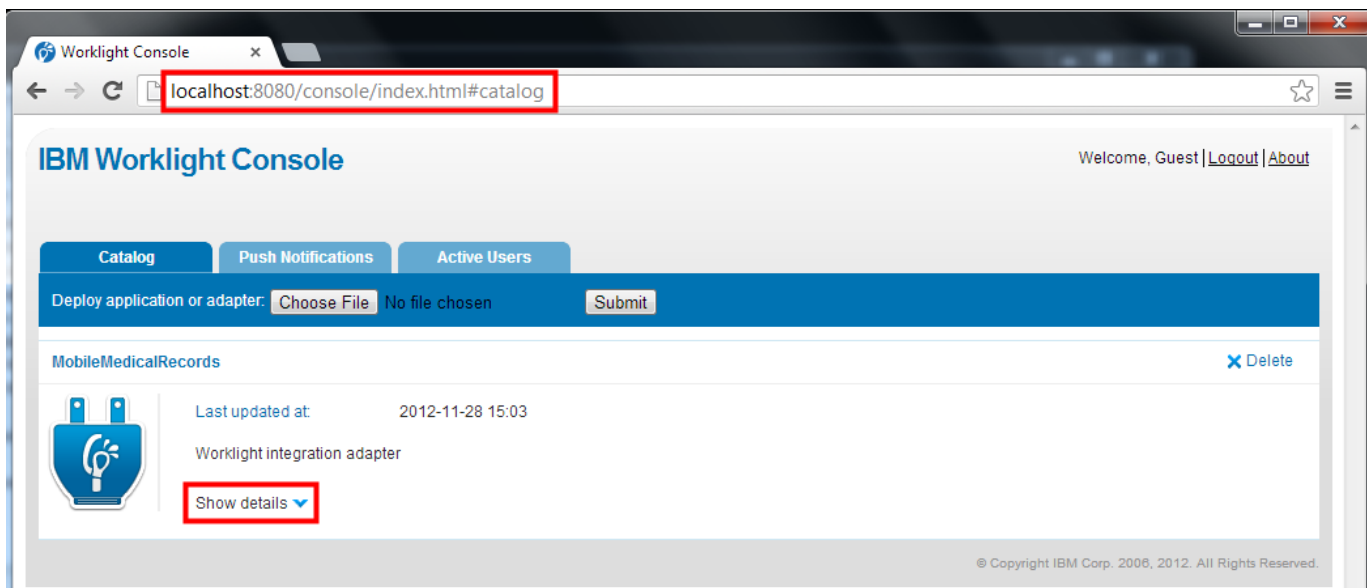
Start a Google Chrome browser session.

__50. Double click the Chrome icon.



__51. Navigate to <http://localhost:8080/console>.

__52. Expand the **Show details** area.



The procedures in the installed adapter should be visible.

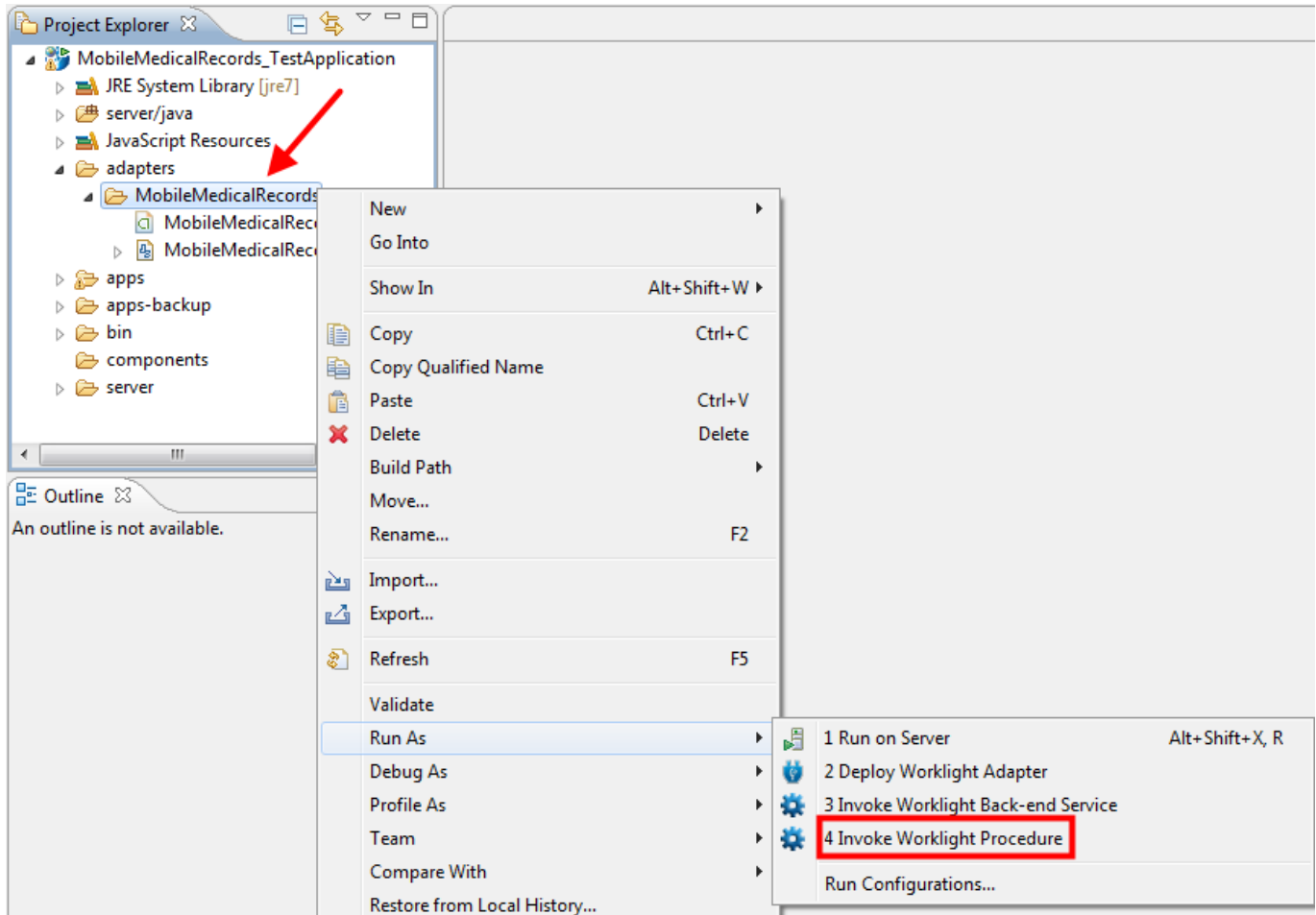
__53. Minimize the browser session.

The screenshot displays the IBM Worklight Console interface. At the top, the title 'IBM Worklight Console' is on the left, and 'Welcome, Guest | [Logout](#) | [About](#)' is on the right. Below the title bar, there are three tabs: 'Catalog', 'Push Notifications', and 'Active Users'. A blue bar contains the text 'Deploy application or adapter: Choose File No file chosen Submit'. The main content area shows the configuration for the 'MobileMedicalRecords' adapter, with a 'Delete' link on the right. On the left is an icon of a blue shield with a white '6' and a lightbulb. To the right of the icon, the text 'Last updated at: 2012-11-28 15:03' and 'Worklight integration adapter' is displayed. Below this is a table of connectivity settings:

Connectivity:	Type:	HTTP
	Protocol:	http
	Domain:	localhost
	Port:	7800
	Use Proxy:	false

Below the table, the 'Procedures' field is highlighted with a red arrow. It contains the text: 'GetPatientHistory, FindPatientMRN, GetVitalSigns, GetMedications, GetLabResults, RequestTest'. At the bottom left of the configuration area is a 'Hide details' link with an upward-pointing arrow. The footer of the console shows the copyright notice: '© Copyright IBM Corp. 2006, 2012. All Rights Reserved.'

- __54. Return to the Worklight studio
- __55. Select the **MobileMedicalRecords** adapter.
- __56. Press the right mouse button.
- __57. Select **Run As→Invoke Worklight Procedure** from the menu.



The Worklight test client should open.

__58. Use the drop-down menu to select **GetLabResults** as the **Procedure name**.

__59. Enter **12345** in the **Parameters** field.

__60. Press the **Run** Button.

The screenshot shows the 'Edit Configuration' dialog box. The title bar says 'Edit Configuration'. Inside, the main heading is 'Edit configuration and launch.'. Below this, the 'Name' field is 'Invoke Procedure MobileMedicalRecords_TestApplication - MobileMedicalRecords'. There is a tab labeled 'Invoke Procedure Data'. Under this tab, there are three dropdown menus: 'Project name:' (MobileMedicalRecords_TestApplication), 'Adapter name:' (MobileMedicalRecords), and 'Procedure name:' (GetLabResults). Below these is a 'Signature:' field with the text 'GetLabResults (MRN)'. Then there is a 'Parameters (comma-separated):' field containing the text '12345'. At the bottom right, there are four buttons: 'Apply', 'Revert', 'Run' (highlighted with a red box), and 'Close'. A help icon (?) is located at the bottom left.

The Worklight test client will invoke the MobileMedicalRecords Worklight adapter, which in turn will invoke the message flow (a service that encapsulates the .Net application).

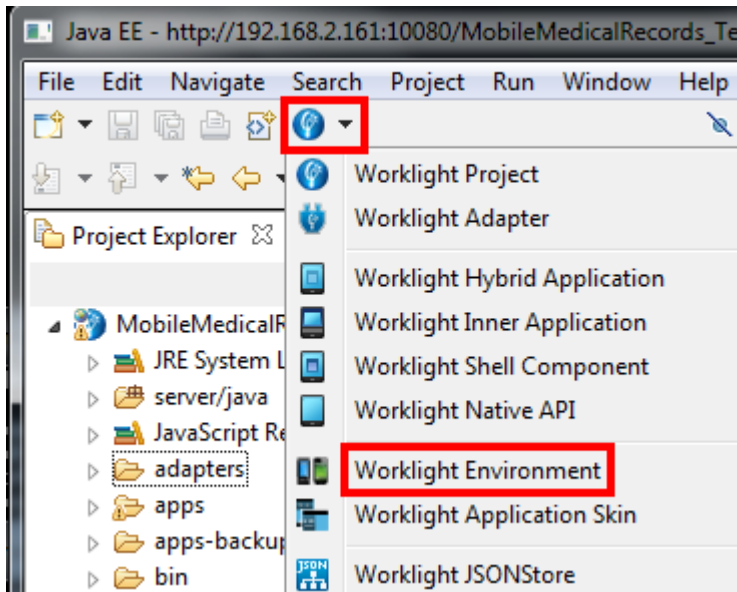
The Test Client will show the value that has been returned from the Integration Bus application when the **GetLabResults** procedure is invoked for patient **12345**. In the example below, a result of **TSH 2.460 uIU/ml for patient Salvatore Monella** was returned.

__61. After examining the results close the **Invoke Procedure Result** window.

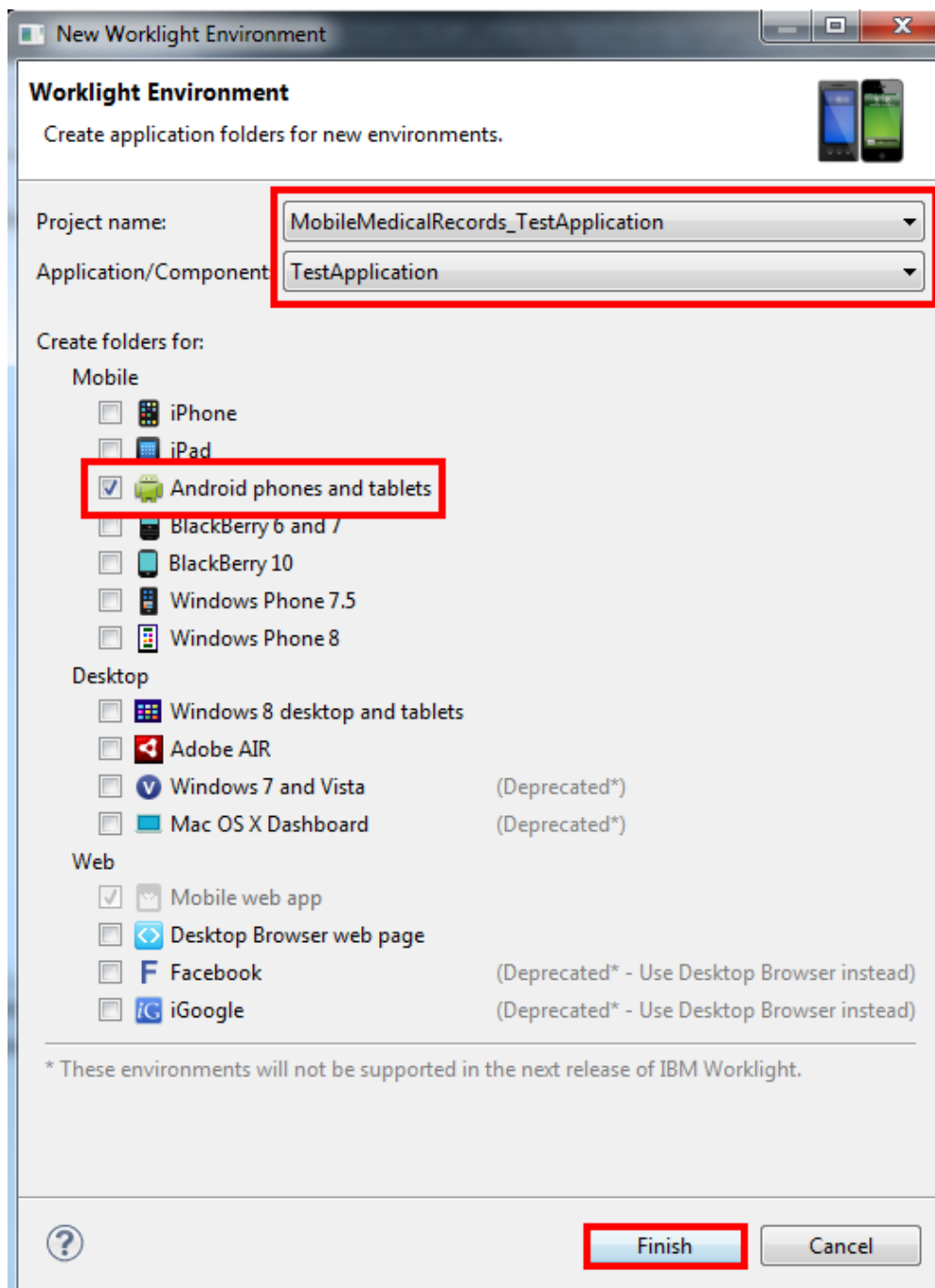


6.4 Test the adapter using the Android emulator

- ___1. Select the down arrowhead next to the Worklight logo on the toolbar.
- ___2. Select **Worklight Environment** from the menu.

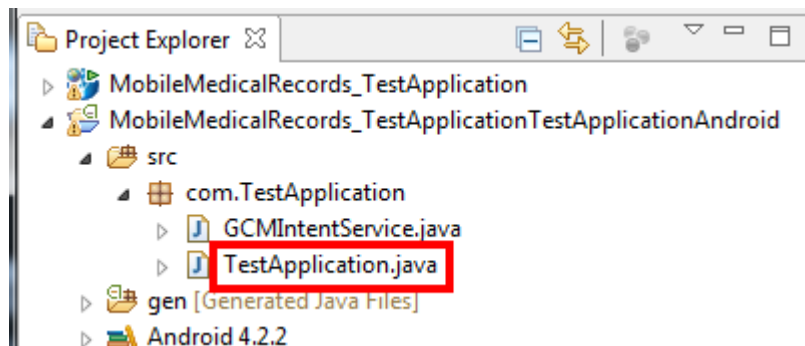


- __3. Use the drop-down menu to select **MobileMedicalRecords_TestApplication** as the **Project name**.
- __4. Use the drop-down menu to select **TestApplication** as the **Application/Component**.
- __5. Select the **Android phones and tablets** check box.
- __6. Press the **Finish** button to create the environment.



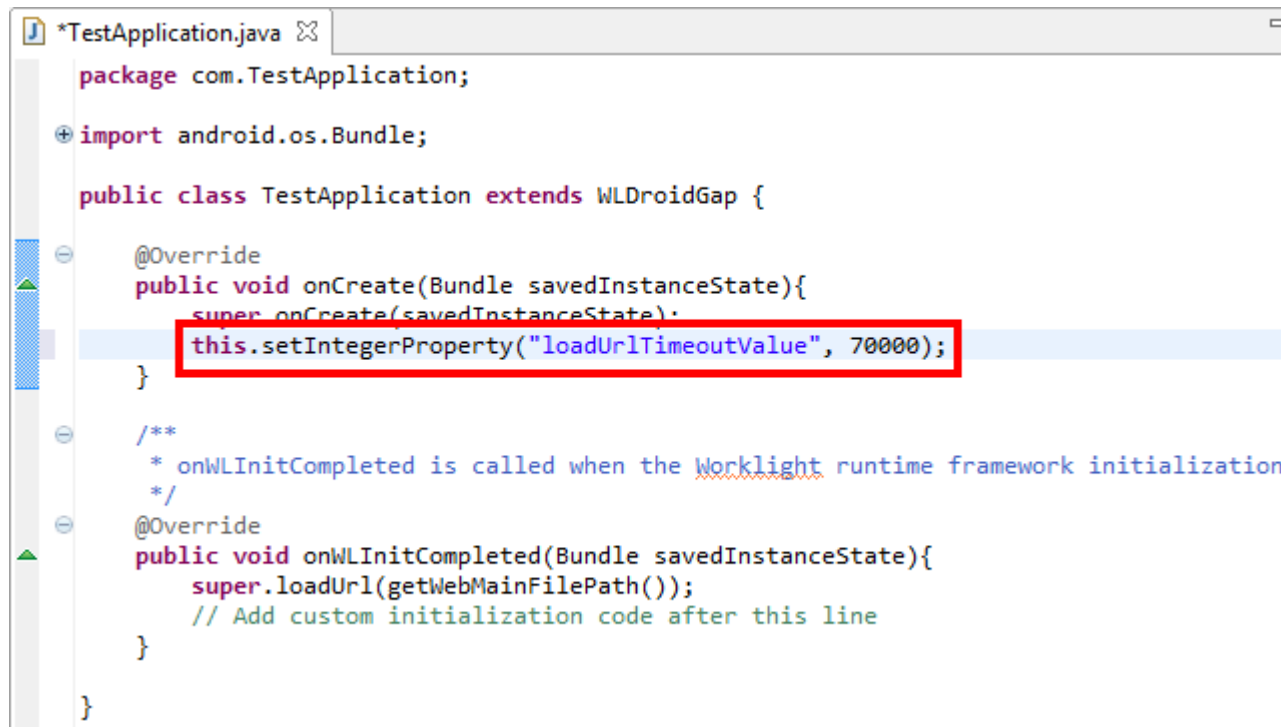
An Android mobile application is generated. The application can require more time to load than the default time out values will permit. This will now be corrected.


- __7. Wait for the new project to be generated.
- __8. Expand the **MobileMedicalRecords_TestApplicationTestApplicationAndroid** folder.
- __9. Expand the **src→com.TestApplication** folders.
- __10. Double-click the **TestApplication.java** application.

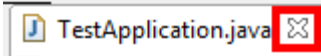


- __11. Insert the following statement as shown in the screen shot below:

```
this.setIntegerProperty("loadUrlTimeoutValue", 70000);
```



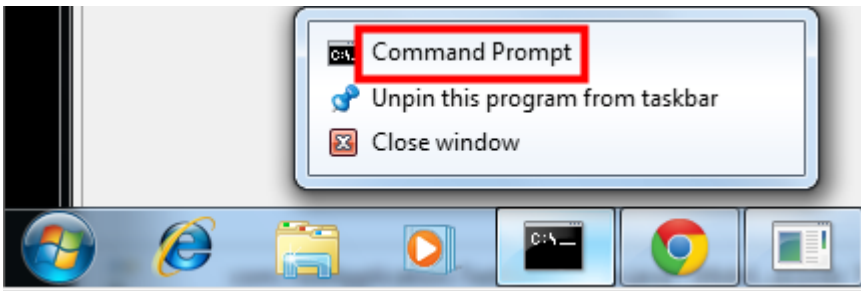
- __12.  Save the modified java program (**Ctrl+S**).
- __13. Close the java editor.



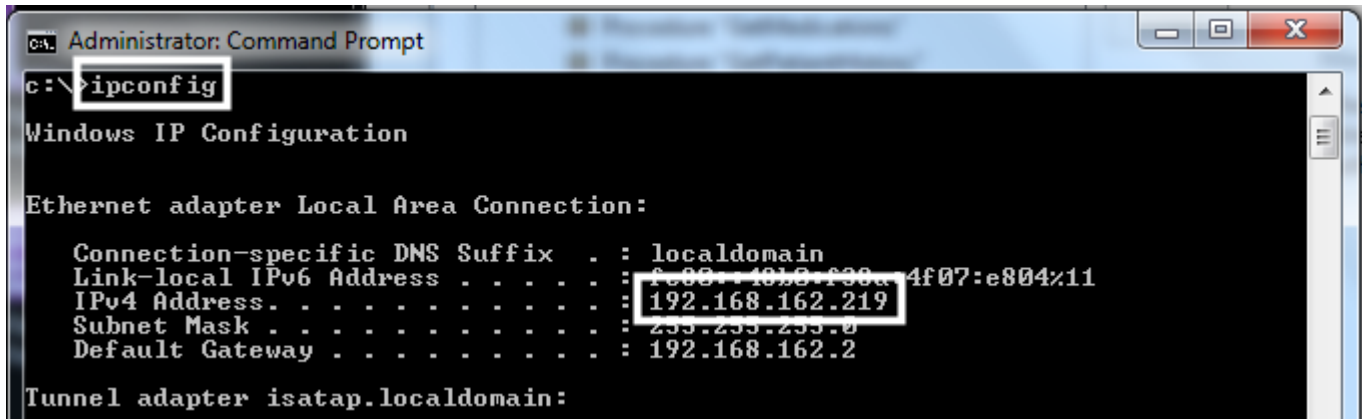
In order to access the adapter from a virtual phone, the location of the test server must be specified. The IP address of the VMWare image must be used. The IP address will likely be different from what is shown in the screen captures in this document.

Start a DOS command prompt.

- __14. Select the DOS command prompt icon.
- __15. Press the right mouse button.
- __16. Select **Command Prompt** from the menu.



- ___17. Execute the **ipconfig** command.
- ___18. Take note of the address of the VMWare image. (**N.B. The address may be different than is shown below.**)



```
Administrator: Command Prompt
c:\>ipconfig

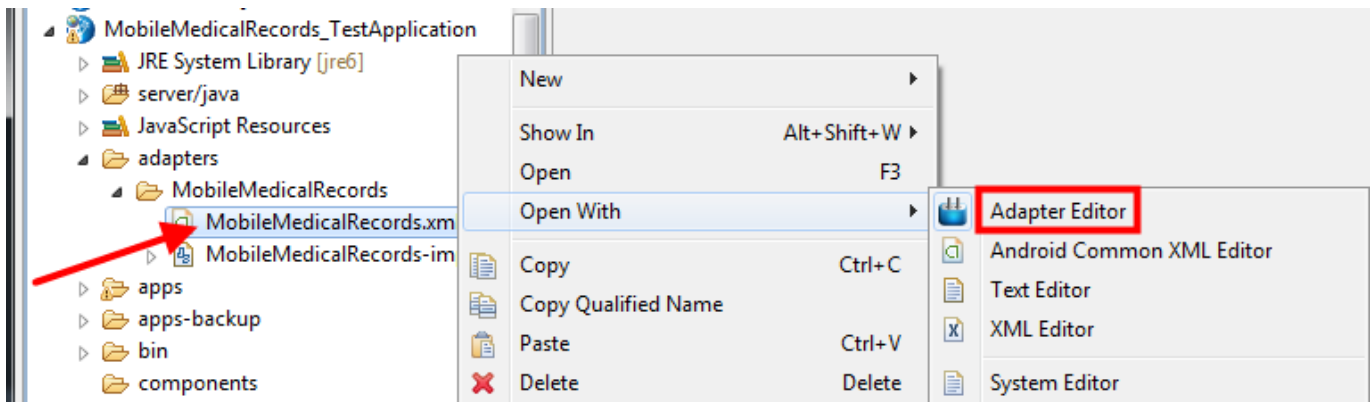
Windows IP Configuration


Ethernet adapter Local Area Connection:

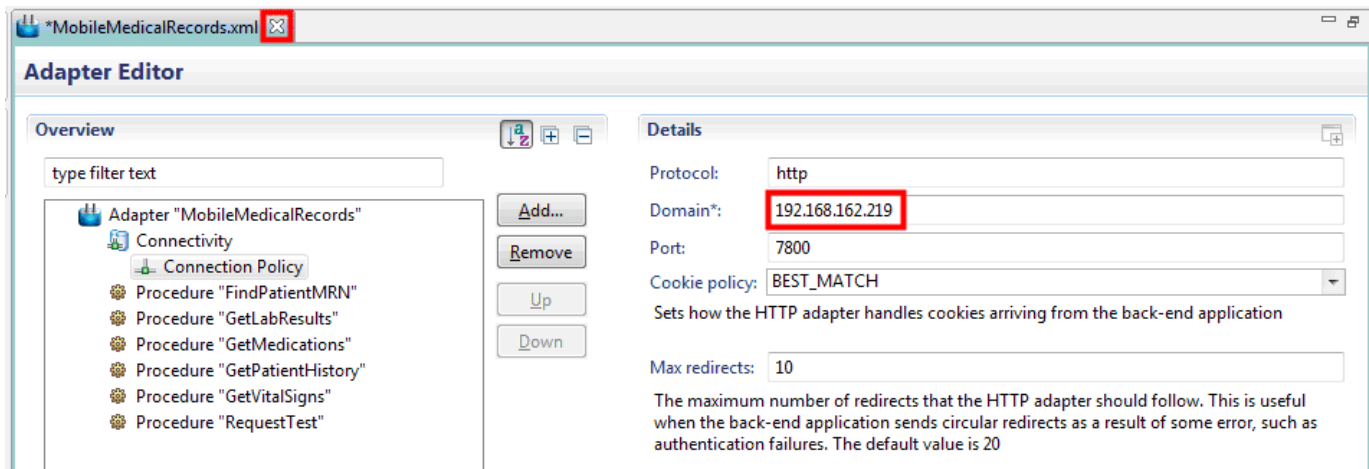
    Connection-specific DNS Suffix  . : localdomain
    Link-local IPv6 Address . . . . . : fe80::1018:f38a:4f07:e804%11
    IPv4 Address. . . . . : 192.168.162.219
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.162.2

Tunnel adapter isatap.localdomain:
```

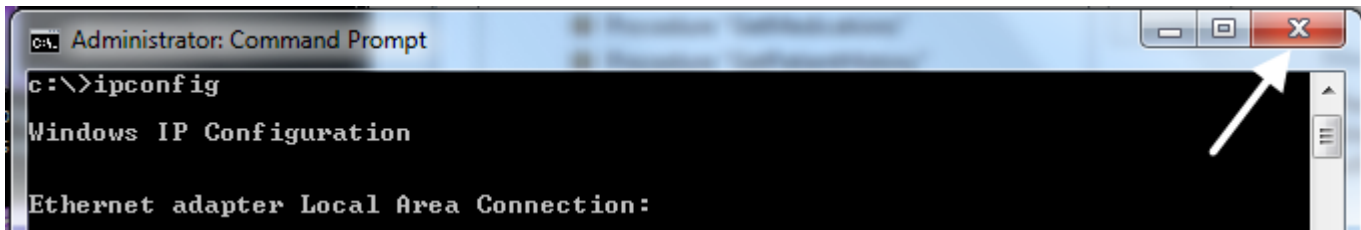
- ___19. In the Worklight Studio, select the **MobileMedicalRecords.xml** adapter specification file.
- ___20. Press the right mouse button.
- ___21. Select **Open With→Adapter Editor** from the menu.



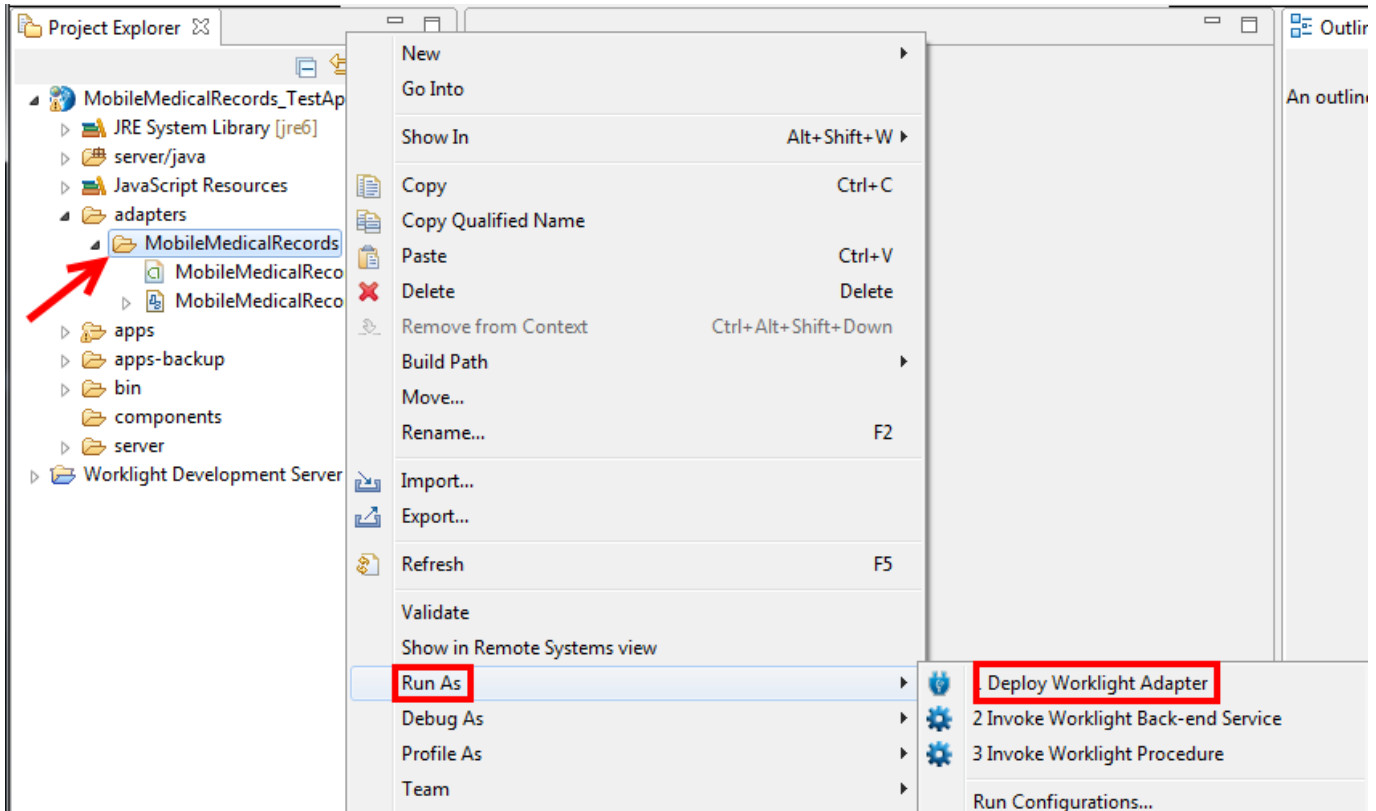
- __22. Expand the **Connectivity** folder.
- __23. Select the **Connection Policy** tab.
- __24. Change the **Domain** address to match the IP address of the VMWare image, as ascertained above.
- __25.  Save the adapter specification.
- __26. Close the adapter editor.



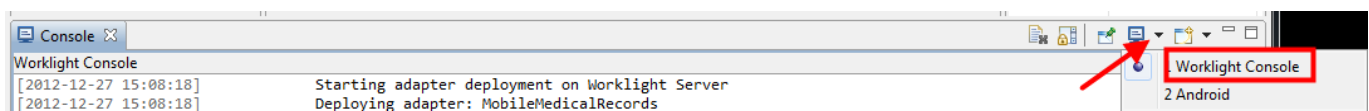
- __27. Close the DOS command prompt.



- __28. Select the **MobileMedicalRecords** adapter.
- __29. Press the right mouse button.
- __30. Select **Run As→Deploy Worklight Adapter** from the menu.

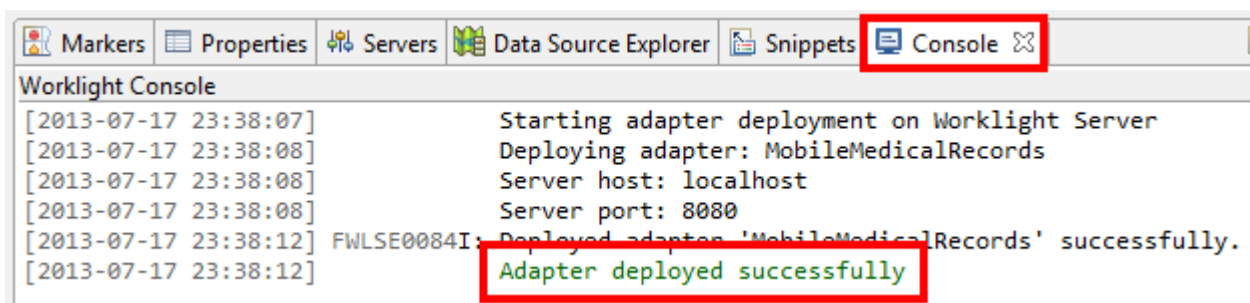


- __31. Use the console icon to select the **Worklight Console** (if necessary).

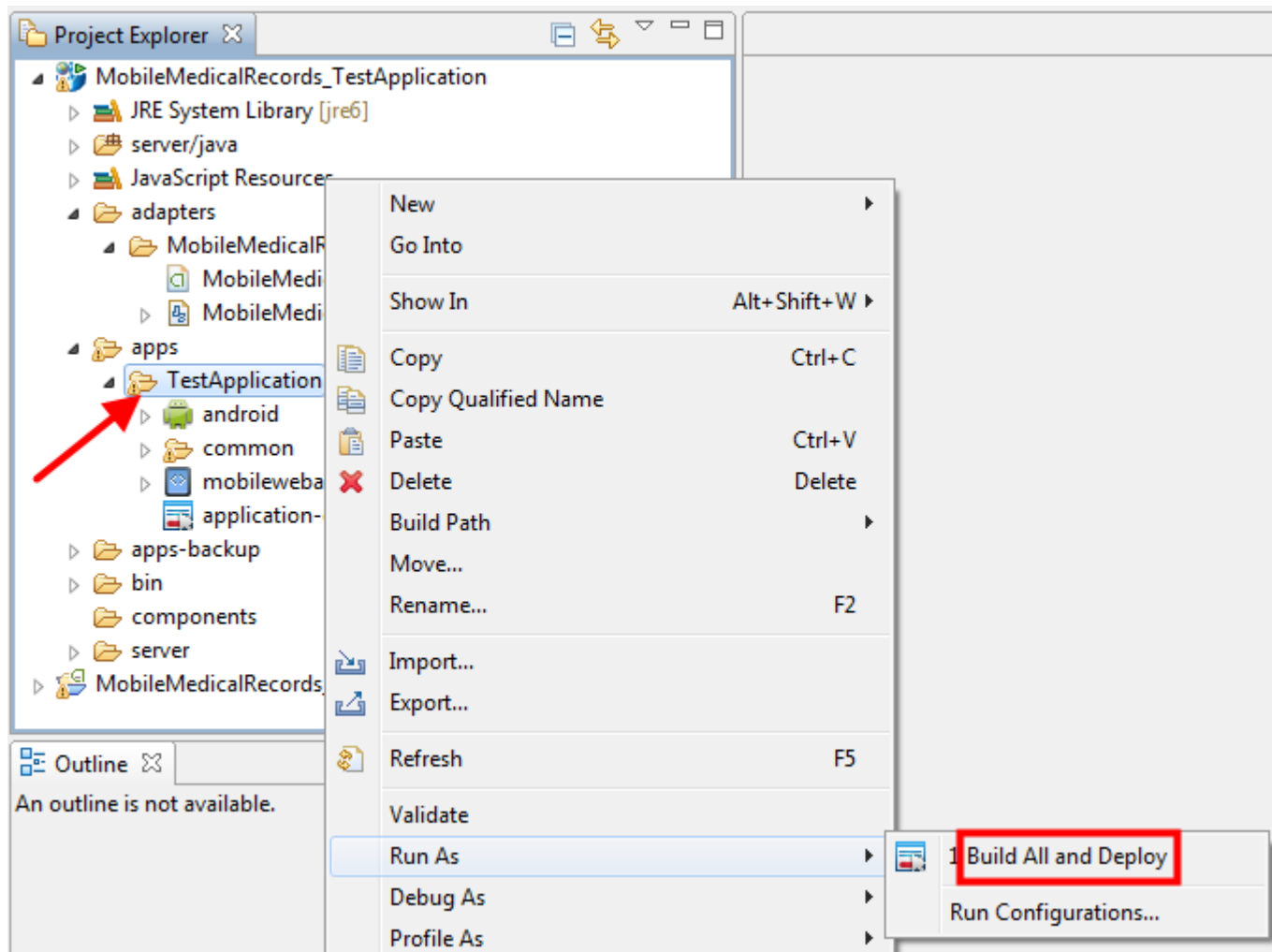


The results of the deployment should be visible in the console.

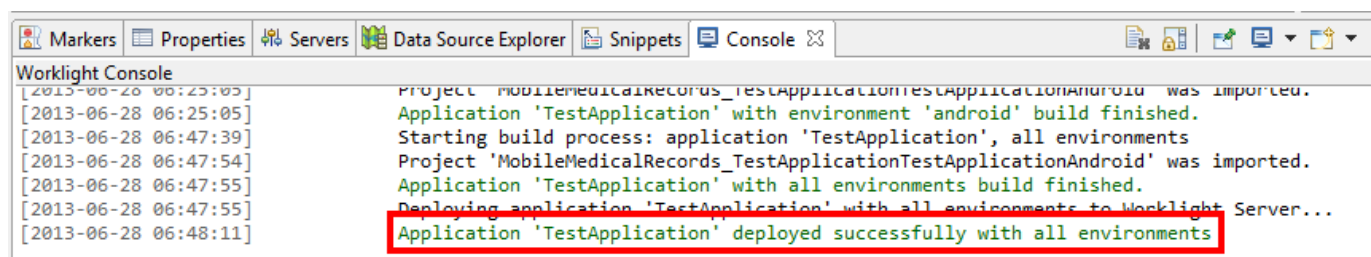
- __32. Confirm that the adapter deployment was successful.



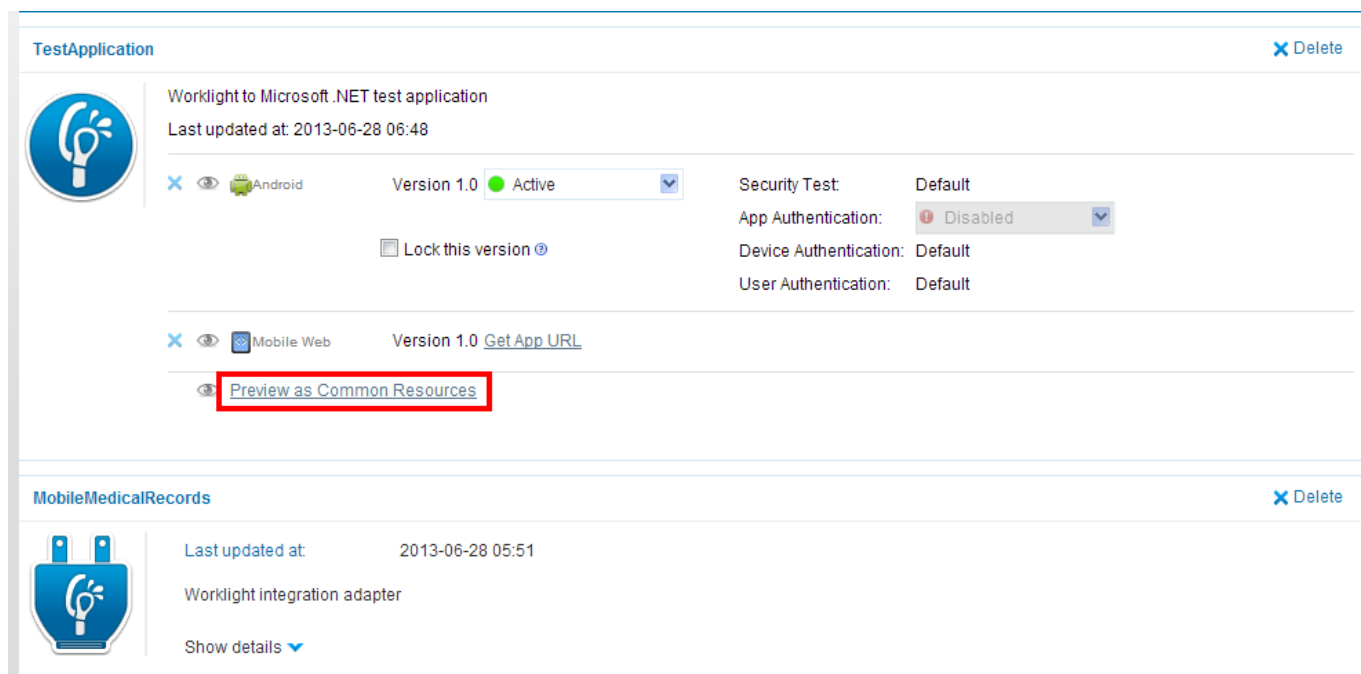
- __33. In the Worklight Studio, expand the **apps** folder.
- __34. Select the **TestApplication** mobile application.
- __35. Press the right mouse button.
- __36. Select **Run As→Build All and Deploy** from the menu.



Wait for the **TestApplication** application to finish building.

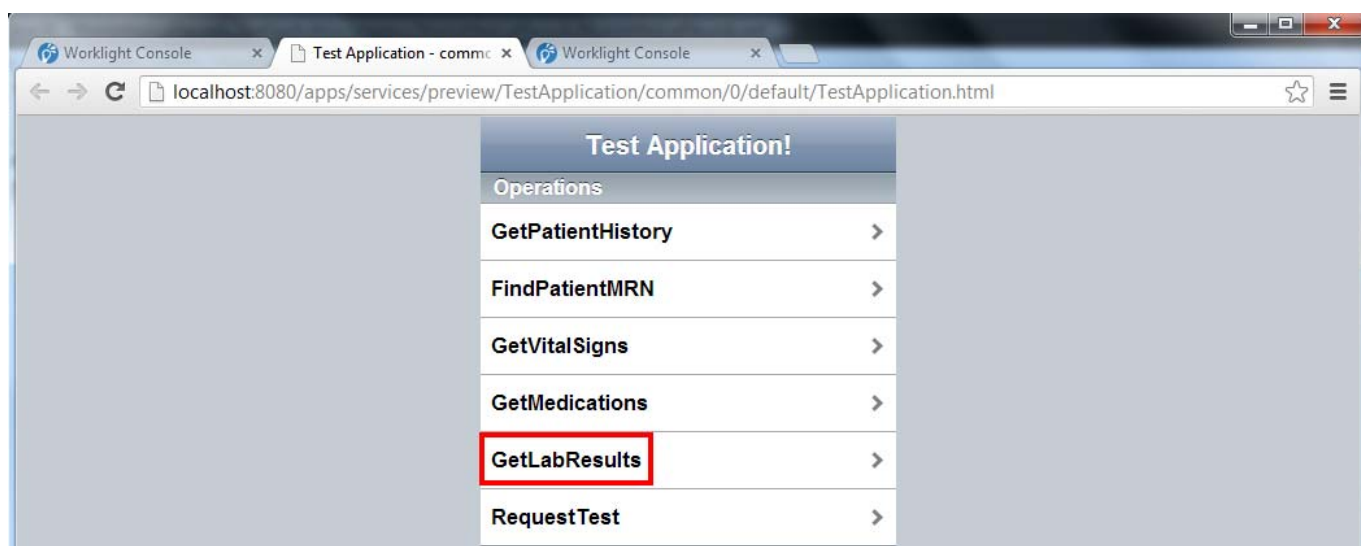


- __37. Return to the Web Browser session.
- __38. Press the **F5** key to refresh the screen.
- __39. The **TestApplication** application should now be visible.
- __40. Click on the **Preview as Common Resources** hot spot.



A new browser tab should open. The HTML version of the application should be visible.

- __41. Select the **GetLabResults** line.

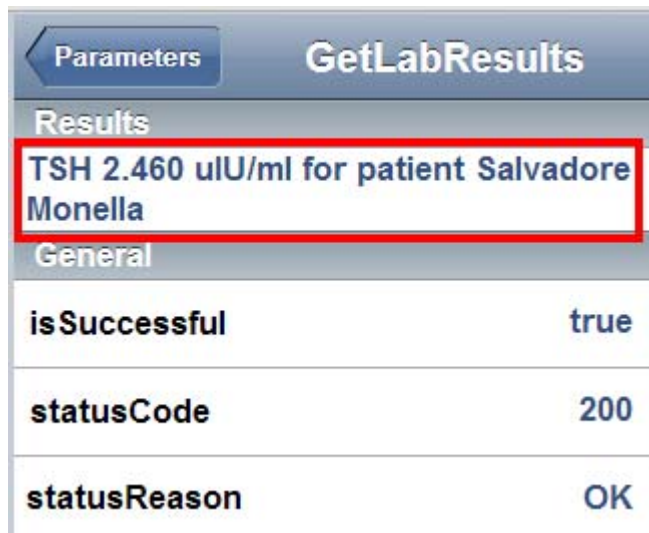


__42. Enter **12345** in the **MRN** field.

__43. Press the **Invoke** button.



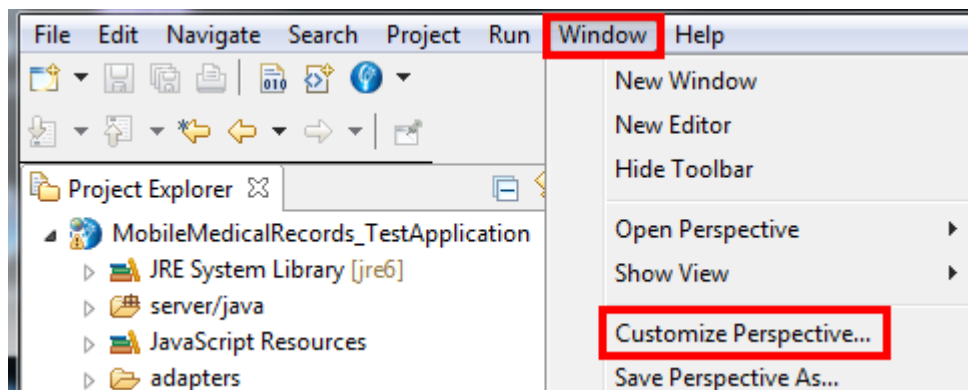
__44. Confirm that the results screen is displayed.



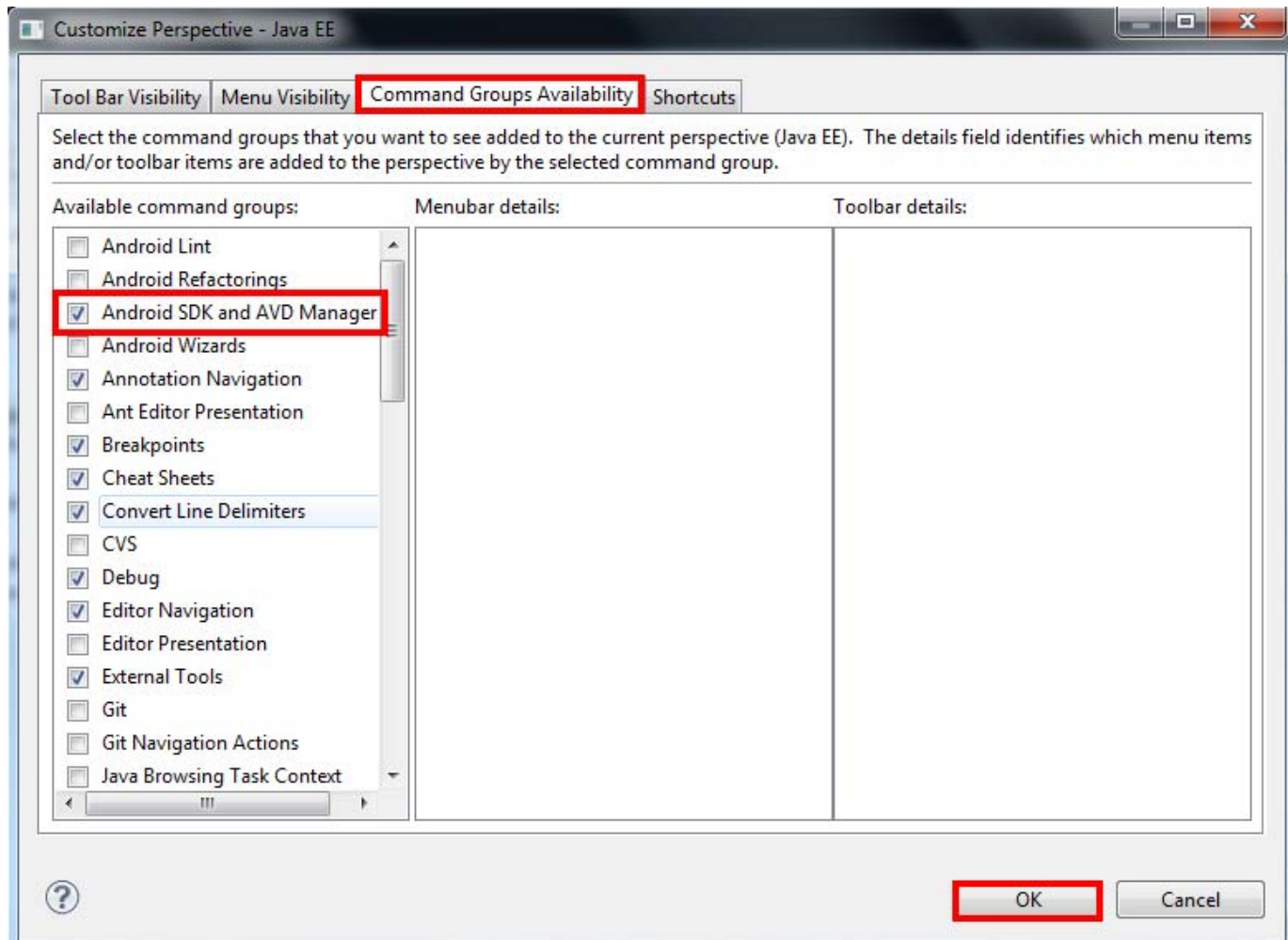
The Android virtual device manager will be added to the toolbar.

__45. Return to the Worklight studio.

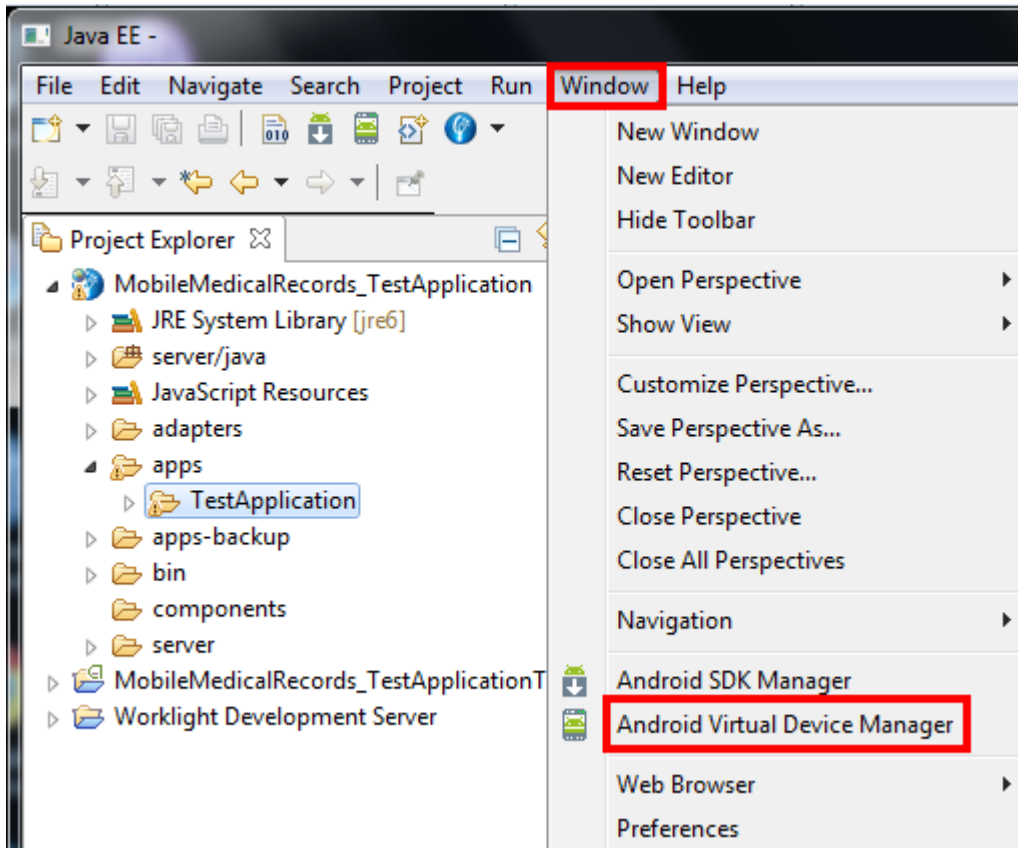
__46. Select **Window**→**Customize Perspective....**



- __47. Select the **Command Groups Availability** tab.
- __48. Select the check box next to **Android SDK and AVD Manager**.
- __49. Press the **OK** button.

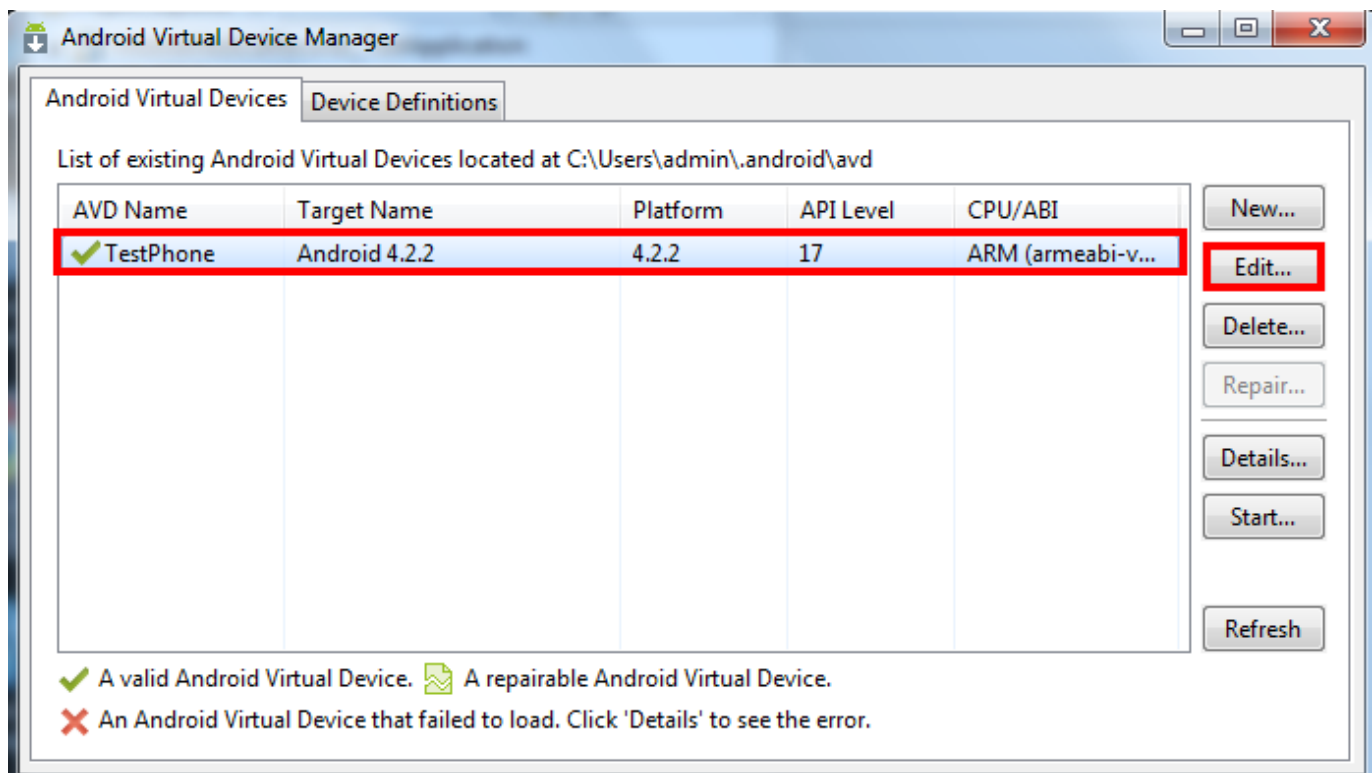


__50. In the Worklight studio select **Window→Android Virtual Device Manager**.



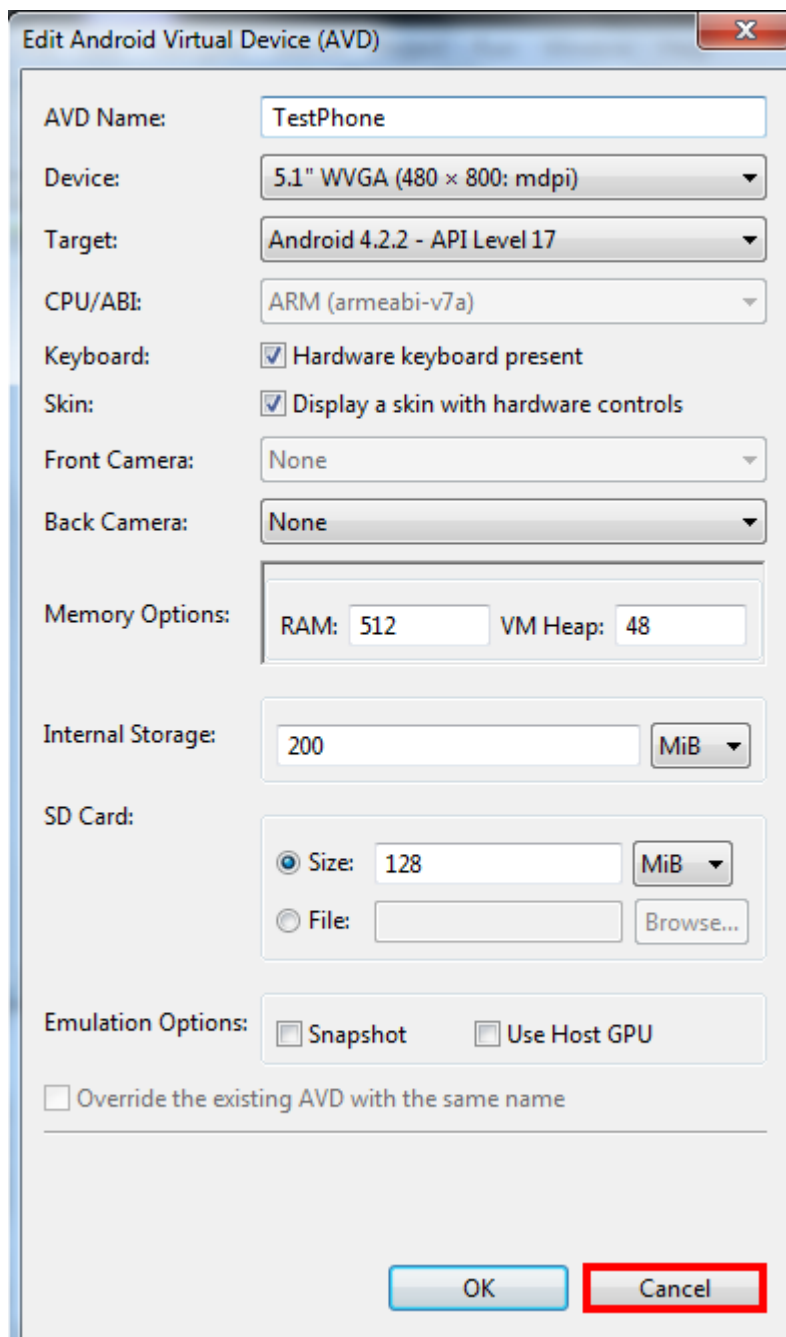
__51. Select the **TestPhone** virtual device.

__52. Press the **Edit** button.

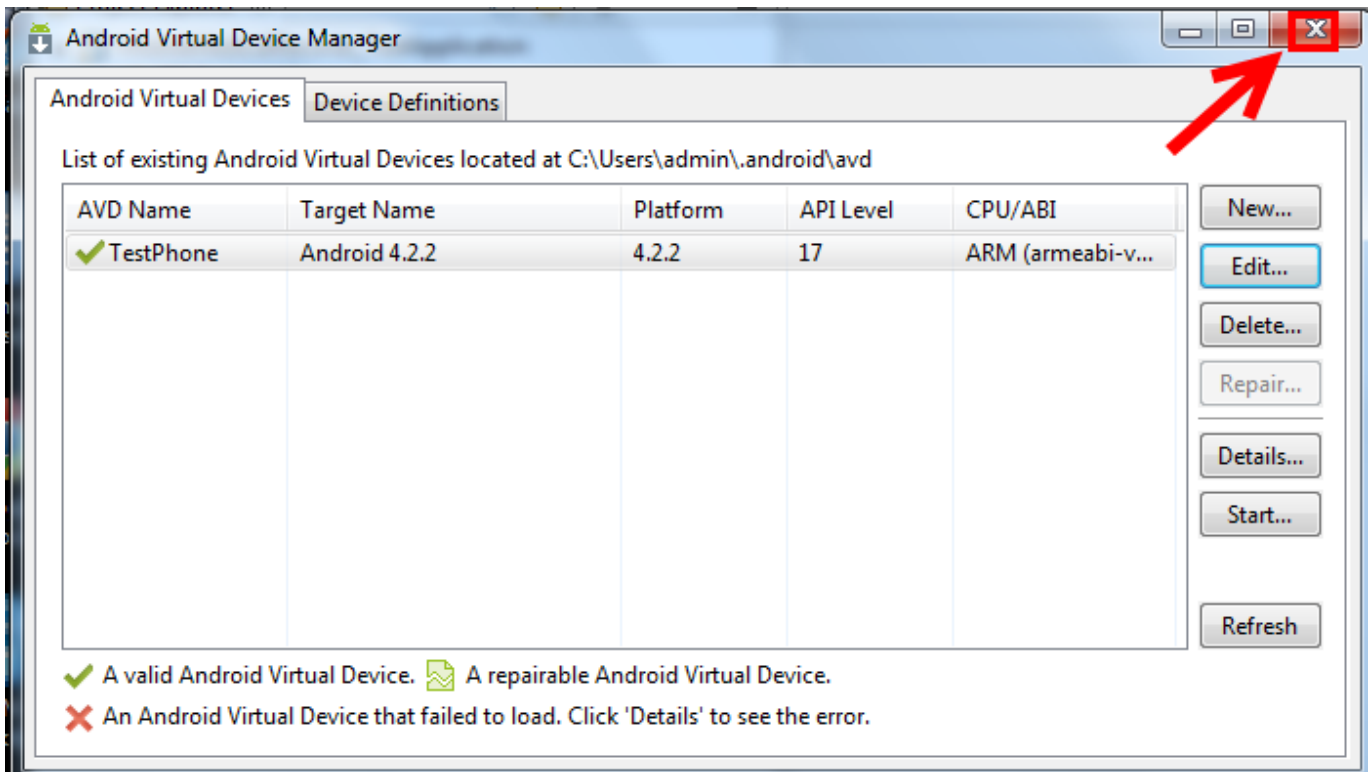


__53. Examine the virtual device. No changes are necessary.

__54. Press the **Cancel** button to exit.



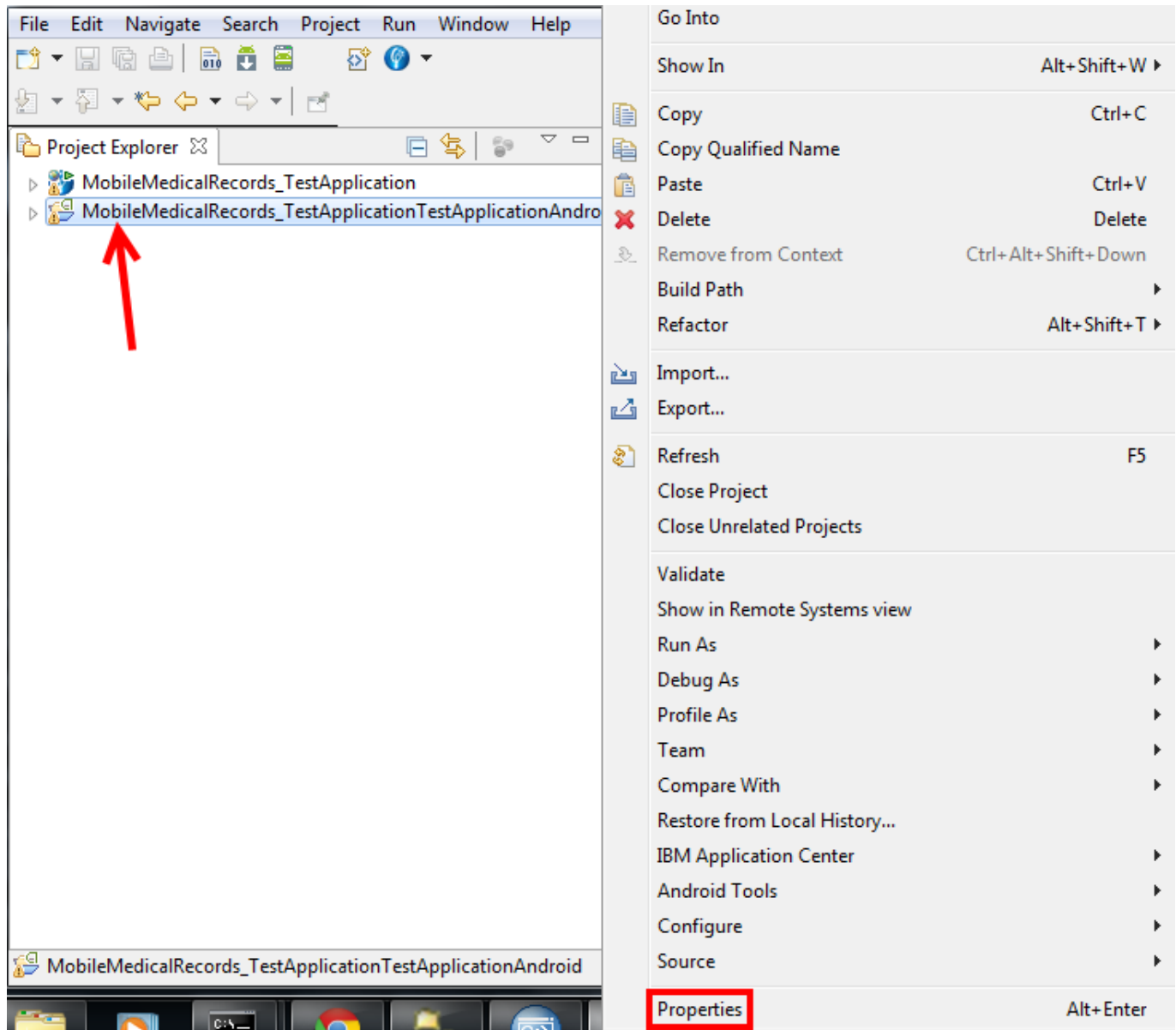
__55. Exit the Android Virtual Device Manager.



__56. Select the **MobileMedicalRecords_TestApplicationTestApplicationAndroid** project.

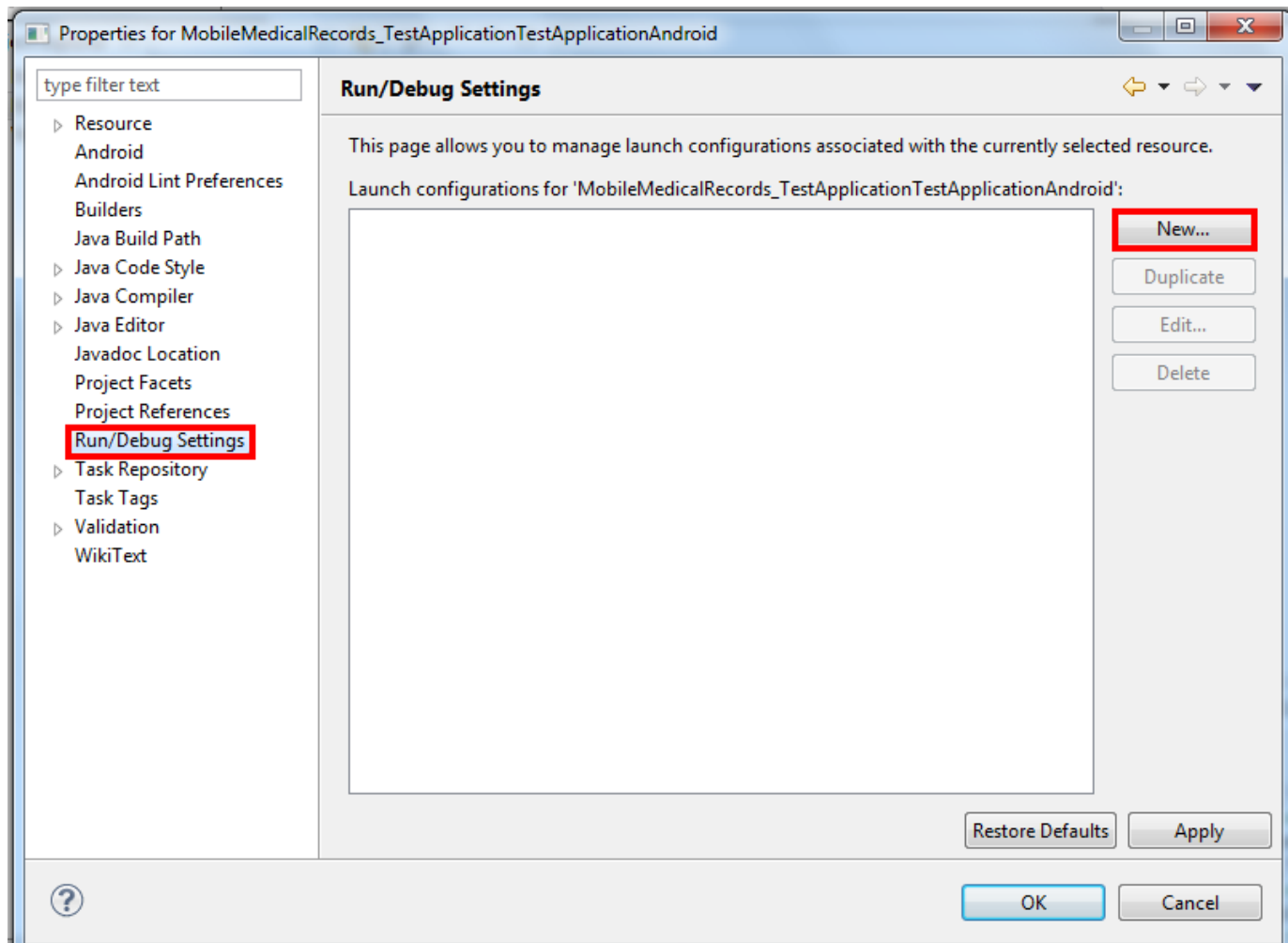
__57. Press the right mouse button.

__58. Select **Properties** from the menu.



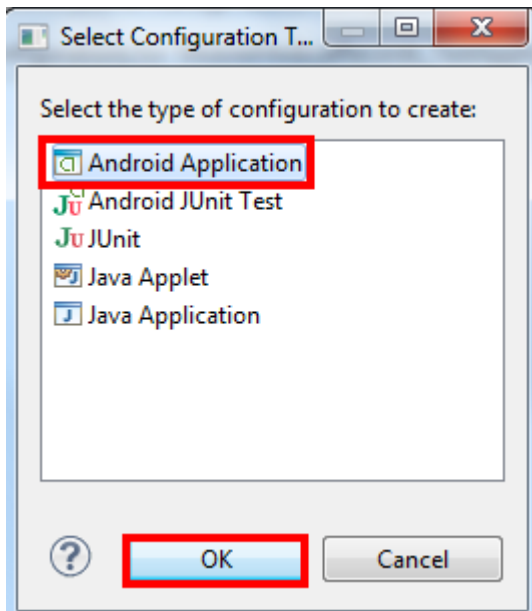
__59. Select the **Run/Debug Settings** tab.

__60. Press the **New** button.



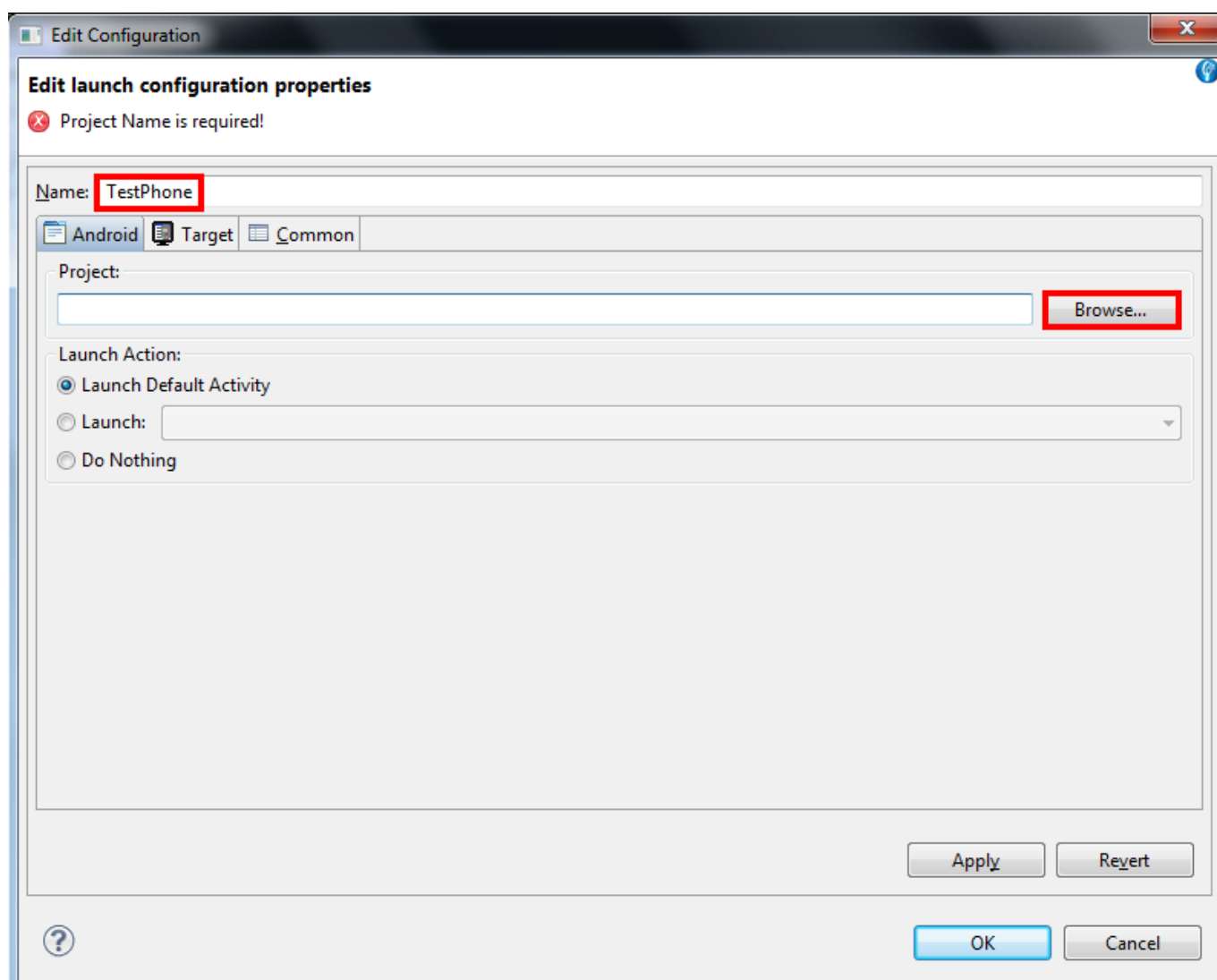
__61. Select **Android Application** as the **type of configuration to create**.

__62. Press the **OK** button to continue.



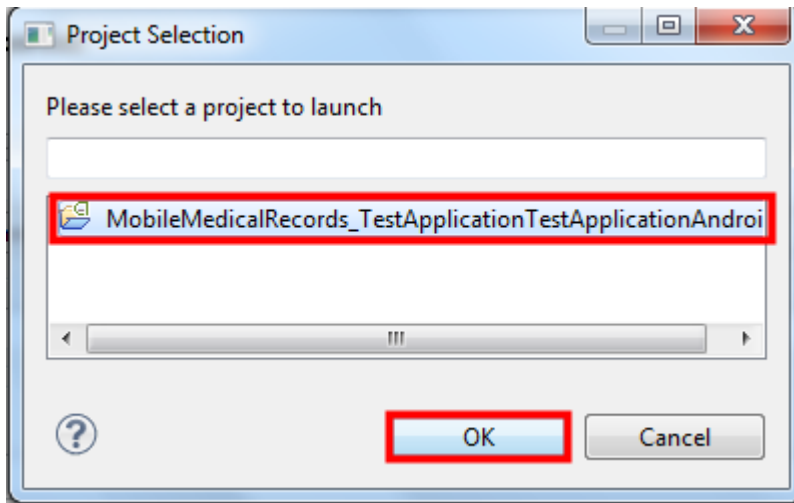
__63. Enter **TestPhone** as the **Name**.

__64. Press the **Browse** button next to the **Project** name field.



__65. Select the **MobileMedicalRecords_TestApplicationTestApplicationAndroid** project.

__66. Press the **OK** button.



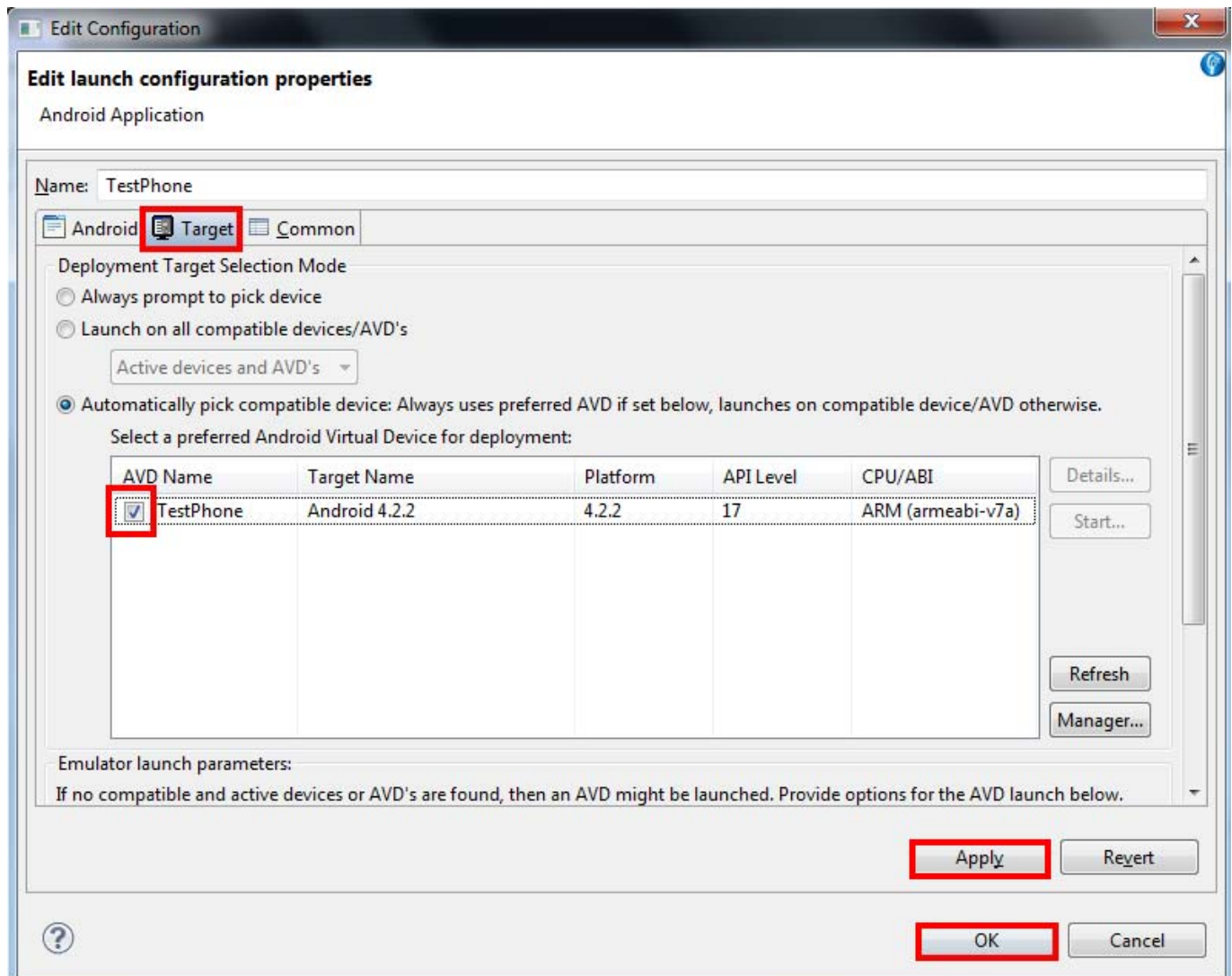
The selected **Project** should be visible.

__67. Select the **Target** tab.

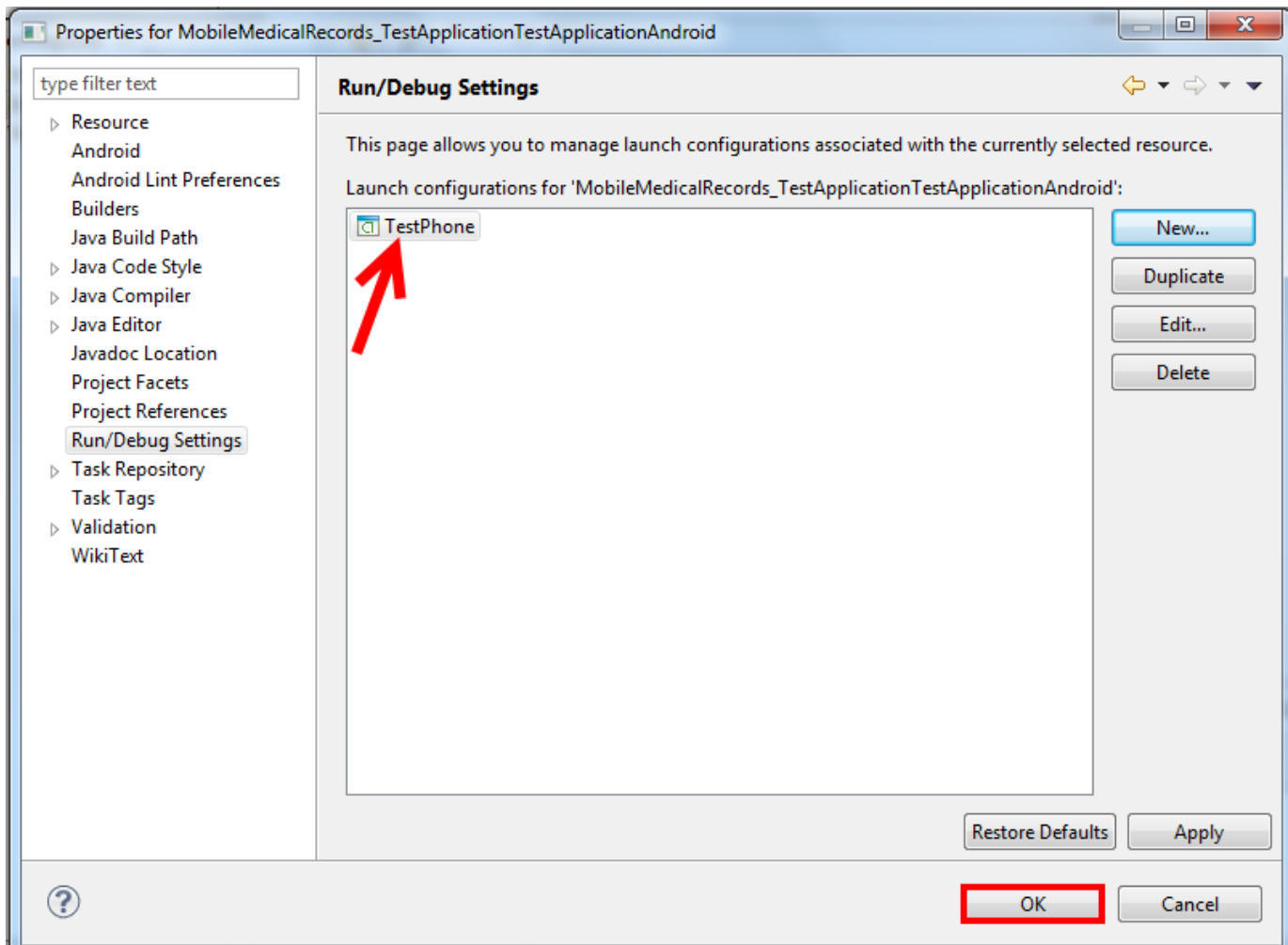
__68. Select the **TestPhone** check box as the **preferred Android Virtual Device for deployment**.

__69. Press the **Apply** button.

__70. Press the **OK** button.



__71. Press the **OK** button.

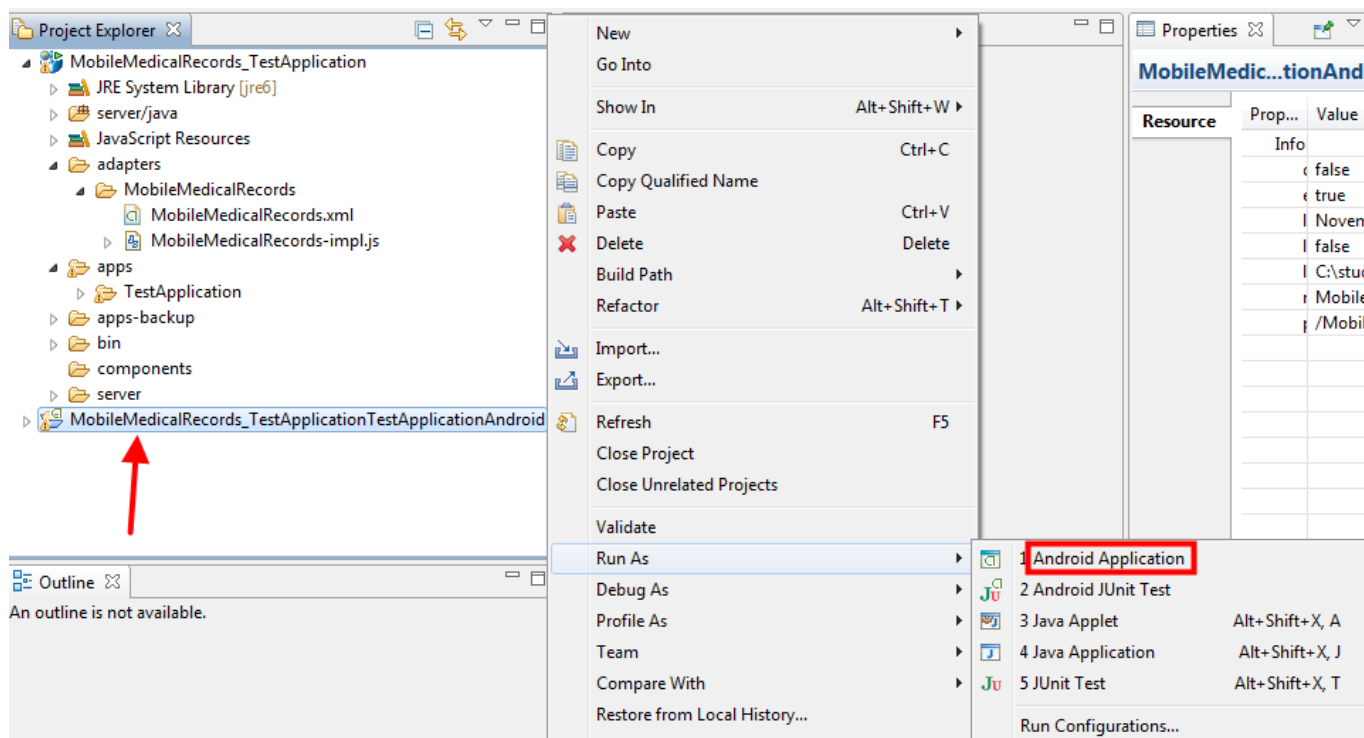


The Android emulator uses a lot of memory. At this point the Integration Toolkit and the Integration Explorer should not be running. If they are running please close them.

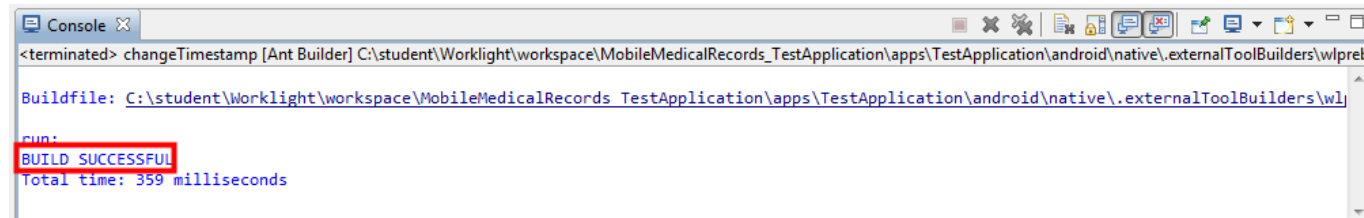
__72. Select the **MobileMedicalRecords_TestApplicationTestApplicationAndroid** project.

__73. Press the right mouse button.

__74. Select **Run As→Android Application** from the menu.



A message should appear on the console confirming that the build of the application was successful.



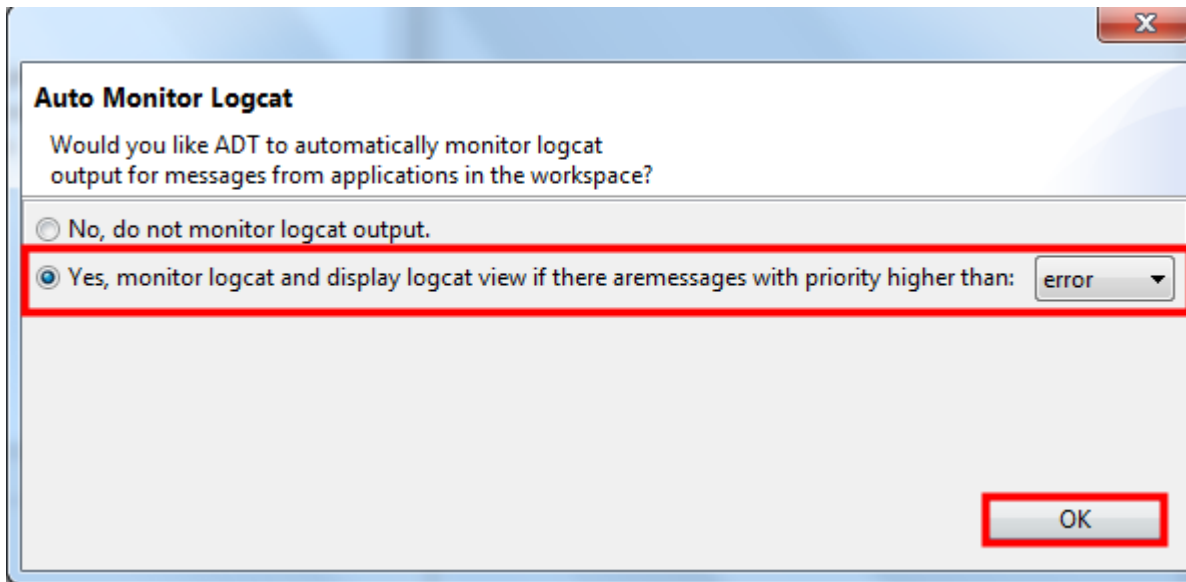
The Android emulator should start. It will take quite a few minutes for Android to boot up. The emulator is large and slow.



Some time after the emulator starts a pop-up dialog should appear. This might appear behind the emulator window.

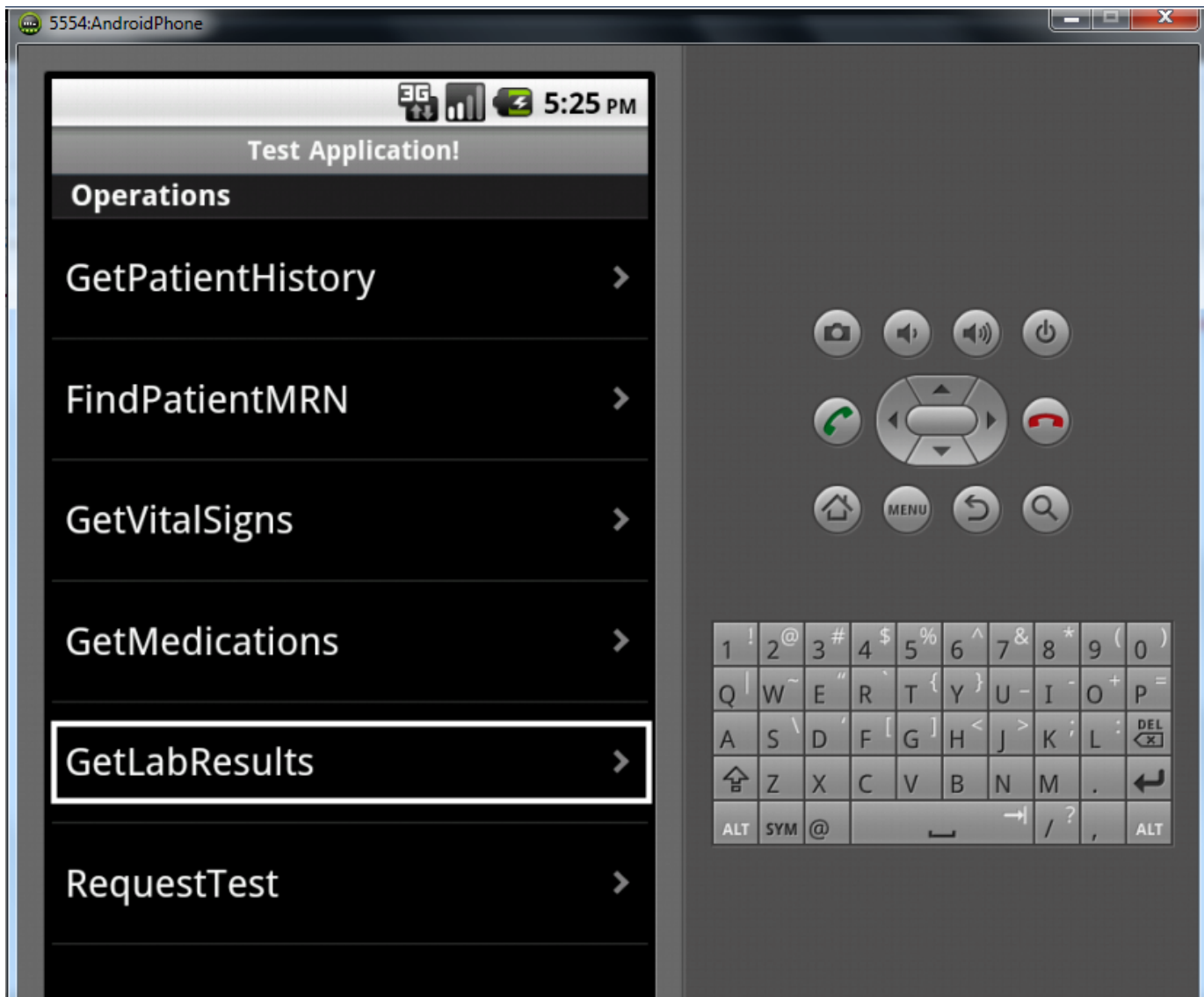
__75. Select the **Yes, monitor logcat and display logcat view ...** radio button.

__76. Press the **OK** button.

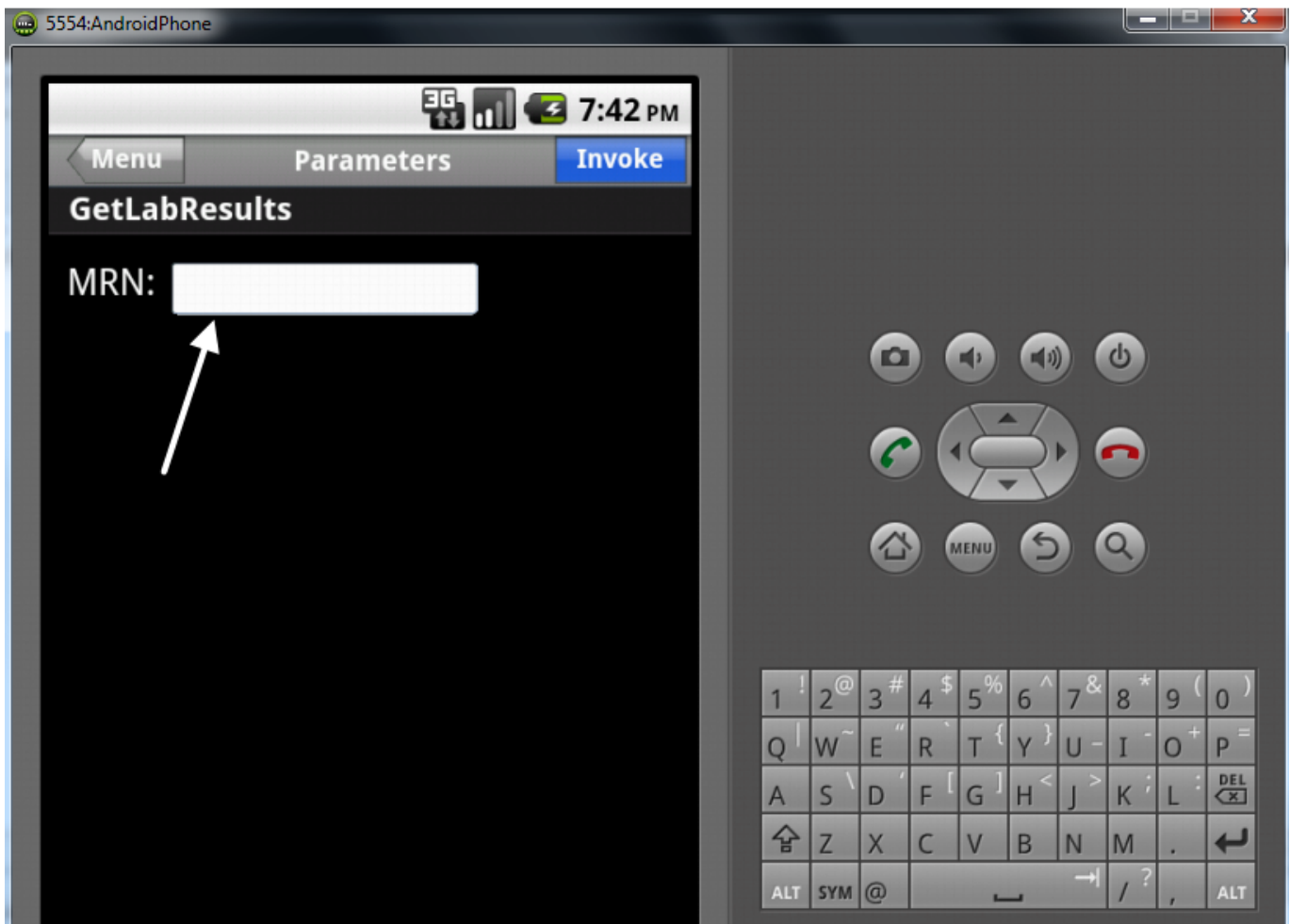


Wait for the emulator to finish starting up. It is large and slow, so please be patient.

__77. Select the **GetLabResults** button.

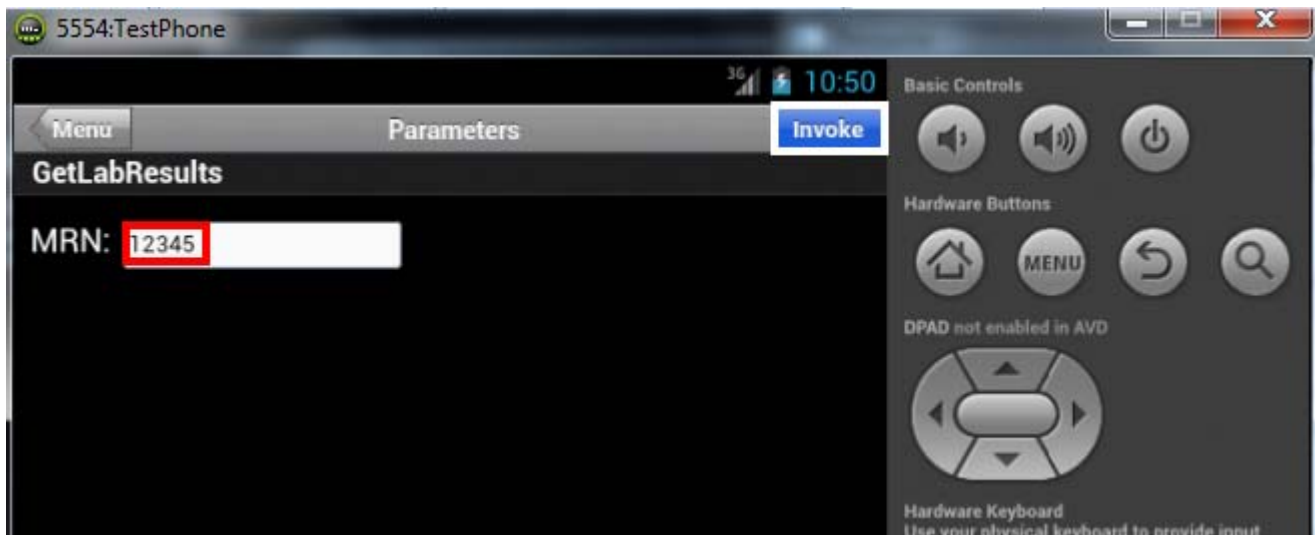


___78. Click in the text box next to **MRN:**



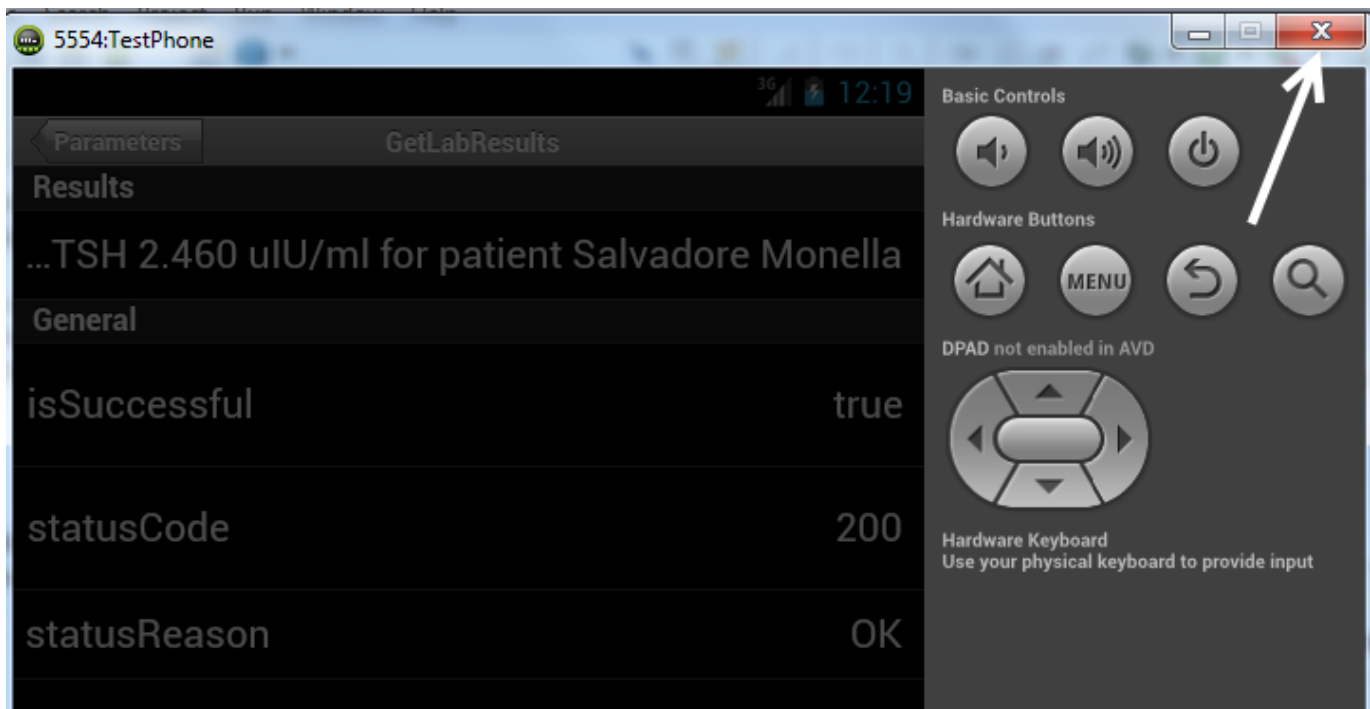
__79. Enter **12345** as the medical record number (**MRN**).

__80. Press the **Invoke** button near the top of the display.



The results will be displayed. If desired return to the main application menu and try some of the other options.

__81. After examining the results the emulator should be closed.



6.5 The Resource handler mobile pattern

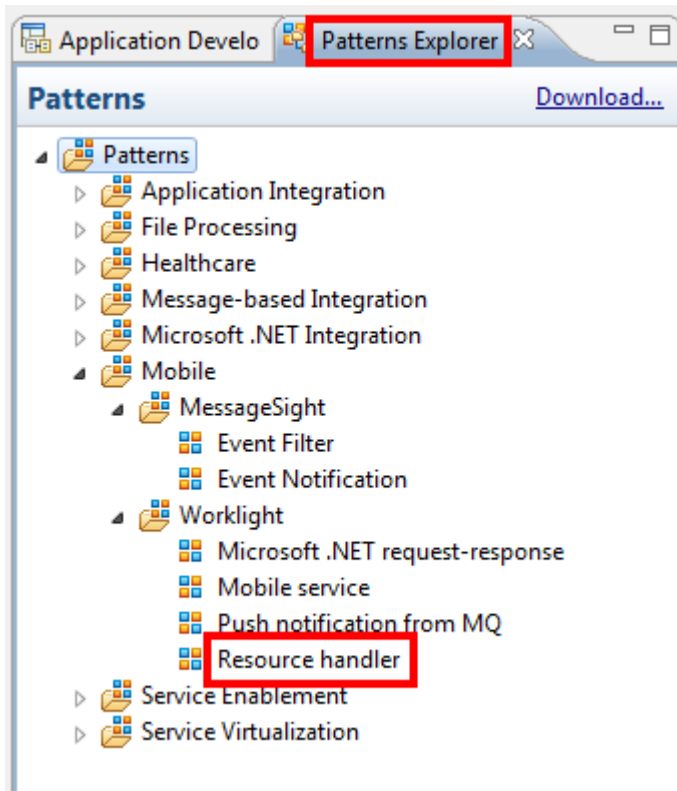
Use this pattern to provide services to mobile applications that use the Worklight APIs. The services are exposed to mobile applications as Worklight adapter procedures that are invoked from the JavaScript in the application.

The pattern is customized by implementing handlers for each of these procedures as subflows in IBM Integration Bus. The pattern provides security policy enforcement and global caching.

The pattern provides a reference implementation in the generated subflows, which can be used to construct a basic mobile example. This section will show you how to use this; specific configurations for user scenarios are left as an exercise for the reader.

This example will use a “Mobile Cars” scenario, where car details can be added to a database (emulated by a shared memory area in Integration Bus). The details can then be retrieved from Integration Bus. The retrieve operation will first attempt to retrieve the information from the global cache. If not available from the cache, the details will be retrieved from the database (shared memory), and added to the cache for future retrievals.

- __1. Return to or start the Integration Toolkit.
- __2. Select the **Patterns Explorer** tab.
- __3. Scroll down so that the **Mobile→Worklight** category is visible.
- __4. Select the **Resource handler** pattern.



Take some time to read about the pattern.

__5. Click **Create New Instance**.

Pattern Specification

View Pattern Specification

View information about the selected pattern and then click the "Create New Instance" button or click [here](#) to start using a pattern.

Worklight: resource handler pattern

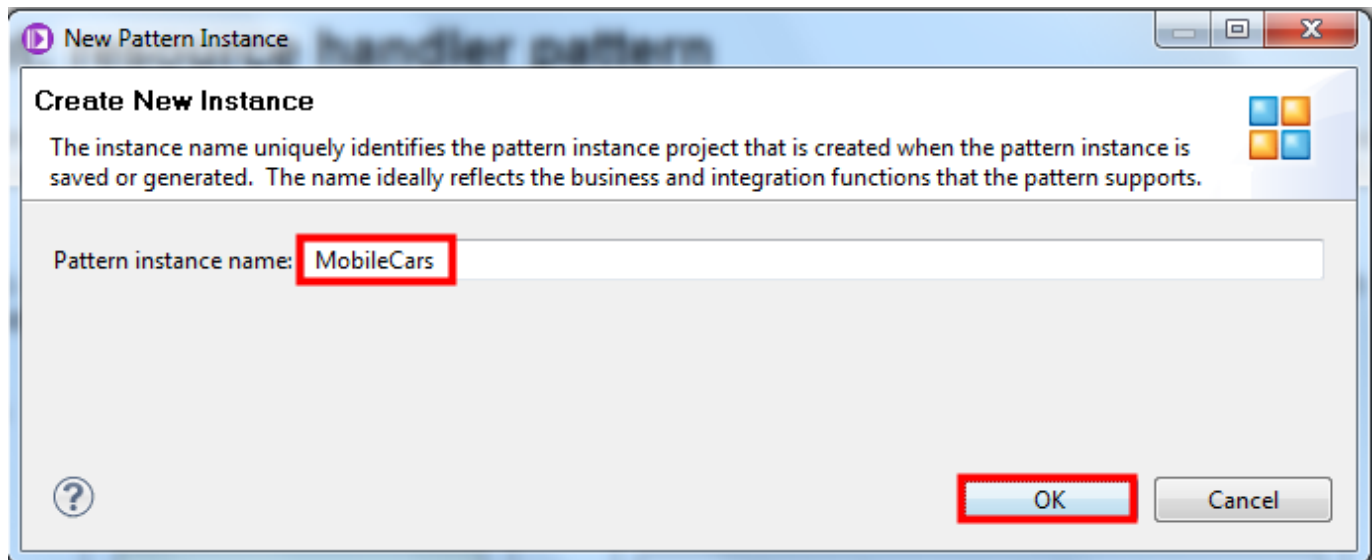
Use this pattern to provide services to mobile applications that use the Worklight APIs. The services are exposed to mobile applications as Worklight adapter procedures that are invoked from the JavaScript in the application.

The pattern is customized by implementing handlers for each of these procedures as subflows in WebSphere Message Broker. The pattern provides security policy enforcement and global caching.

The diagram illustrates the Worklight Resource Handler Pattern. On the left, four mobile device icons are shown. An arrow labeled 'REST (JSON/HTTP)' points from these devices to a box labeled 'Worklight'. Inside the 'Worklight' box is a 'Mobile Application' box containing 'Javascript Procedures' and a 'Message Broker Adapter (CRUD)'. An arrow points from the 'Message Broker Adapter (CRUD)' to a 'WebSphere Message Broker' box. Inside this box is a 'Worklight Resource Handler Pattern Instance' box containing a 'Resource Handler' box. The 'Resource Handler' is connected to a 'WebSphere Extreme Scale Cache' cloud, which has a 'Read' operation. Below the 'Resource Handler' are three buttons: 'Create', 'Update', and 'Delete'. At the bottom, there are two lock icons labeled 'Authentication' and 'Authorization'.

Create New Instance

- __6. Enter **MobileCars** as the **Name**.
- __7. Press the **OK** button to generate the pattern instance.



The pattern parameters editor should open.

__8. Expand the **Worklight** section.

__9. Remove the selection from the **Enable audit** check box.

The screenshot shows the 'Pattern Parameters' editor. The 'Worklight' section is expanded, showing fields for 'Worklight version' (Worklight v5.0), 'Adapter description' (Worklight integration adapter), 'Maximum concurrent connections *' (99), and 'Enable audit *' (unchecked). The 'Service Configuration' section is collapsed. The 'Pattern Parameters Details' pane on the right shows a list of sections: Worklight, Service Configuration, Security (LDAP), Logging, Error handling, and General.

__10. Collapse the **Worklight** section

__11. Expand the **Service Configuration** section.

__12. Enter **Car** as the **Resource name**.

__13. Select the **Cache results** check box.

The screenshot shows the 'Pattern Parameters' editor. The 'Service Configuration' section is expanded, showing fields for 'Resource name *' (Car), 'Resource operations *' (Full control (CRUD)), 'Cache results *' (checked), and 'Server address *' (http://localhost:7080). The 'Worklight' section is collapsed. The 'Pattern Parameters Details' pane on the right shows a list of sections: Worklight, Service Configuration, Security (LDAP), Logging, Error handling, and General. At the bottom, there is a 'Generate' button and tabs for 'Specification' and 'Configuration'.

__14. Collapse the **Service Configuration** category.

__15. Expand the **Logging** category.

__16. Select the **Logging required** check box.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

- Worklight ☒
- Service Configuration ☒
- Security (LDAP) ☒
- Logging ☒
Enable logging messages and specify their destination
Logging required * ☒
Log queue manager
Log queue *
- Error handling ☒
- General ☒

Generate

Specification Configuration

Pattern Parameters Details

- Worklight
- Service Configuration
- Security (LDAP)
- Logging
- Error handling
- General

__17. Collapse the **Logging** category.

__18. Expand the **General** category.

__19. Enter **CARS.** as the **Queue prefix**. (N.B. remember the period at the end of the prefix.)

__20. Click **Generate** to generate the pattern.

Configure Pattern Parameters

Provide values for pattern parameters. Click the "Generate" button or click [here](#) to generate a pattern instance.

Pattern parameters are ready. Click the "Generate" button to generate a pattern instance.

Pattern Parameters

- Security (LDAP) ☒
- Logging ☒
- Error handling ☒
- General ☒
Specify naming conventions and description
Flow prefix
Flow suffix
Queue prefix
Queue suffix
Short description
Long description

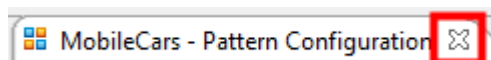
Generate

Specification Configuration

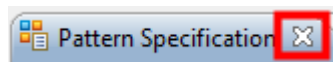
Pattern Parameters Details

- Worklight
- Service Configuration
- Security (LDAP)
- Logging
- Error handling
- General

__21. Close the **MobileCars – Pattern Configuration** tab.

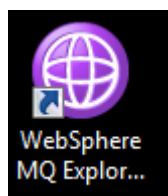


__22. Close the **Pattern Specification** tab.



The message flows use a couple of MQ queues. The queues will be created next.

__23. Either return to or start MQ Explorer.

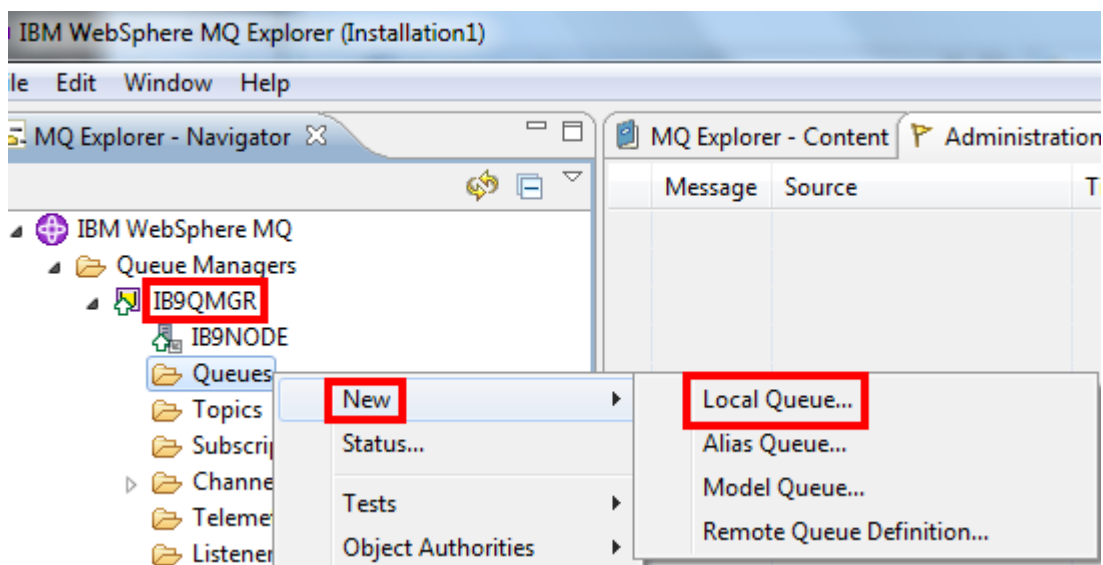


__24. Expand **Queue Managers**→**IB9QMGR**.

__25. Select the **Queues** folder.

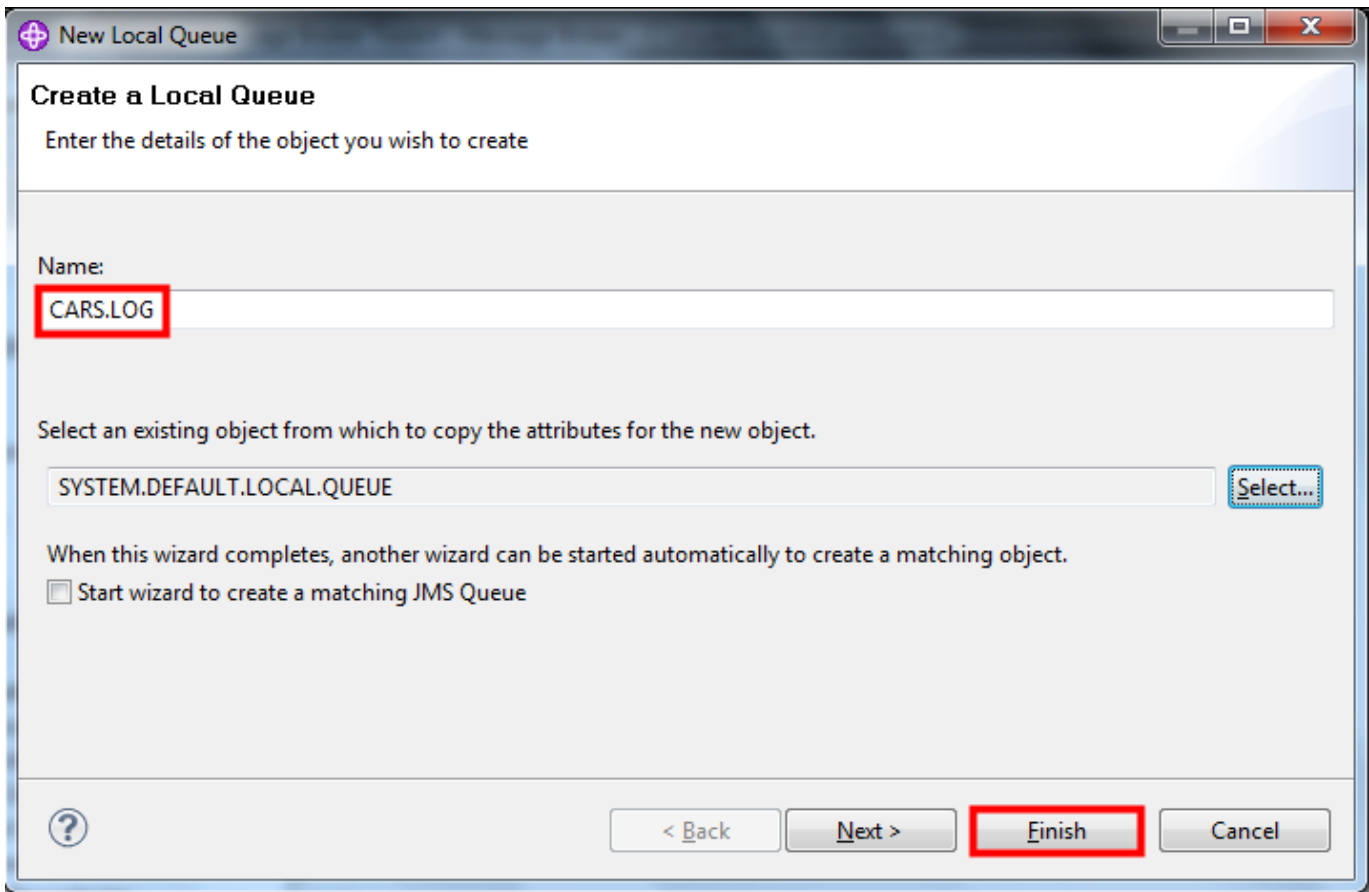
__26. Press the right mouse button.

__27. Select **New**→**Local Queue** from the menu.

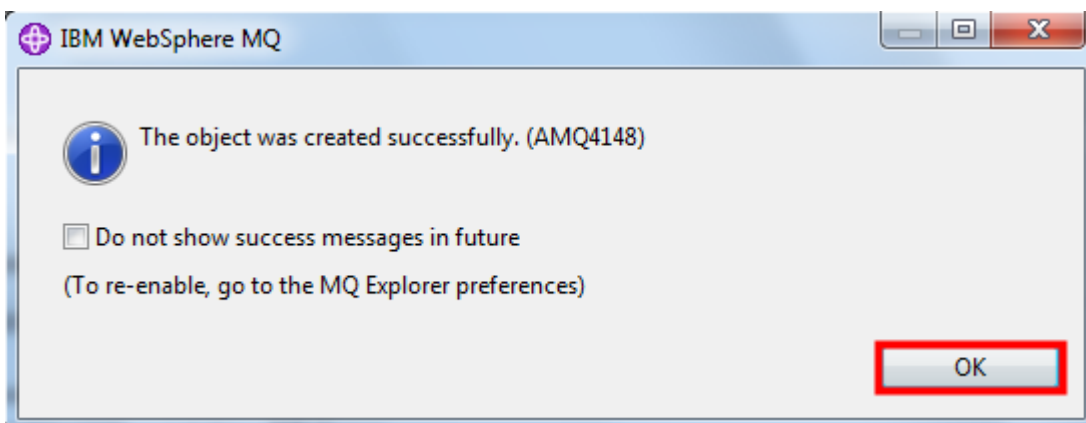


__28. Enter **CARS.LOG** as the **Name** of the queue.

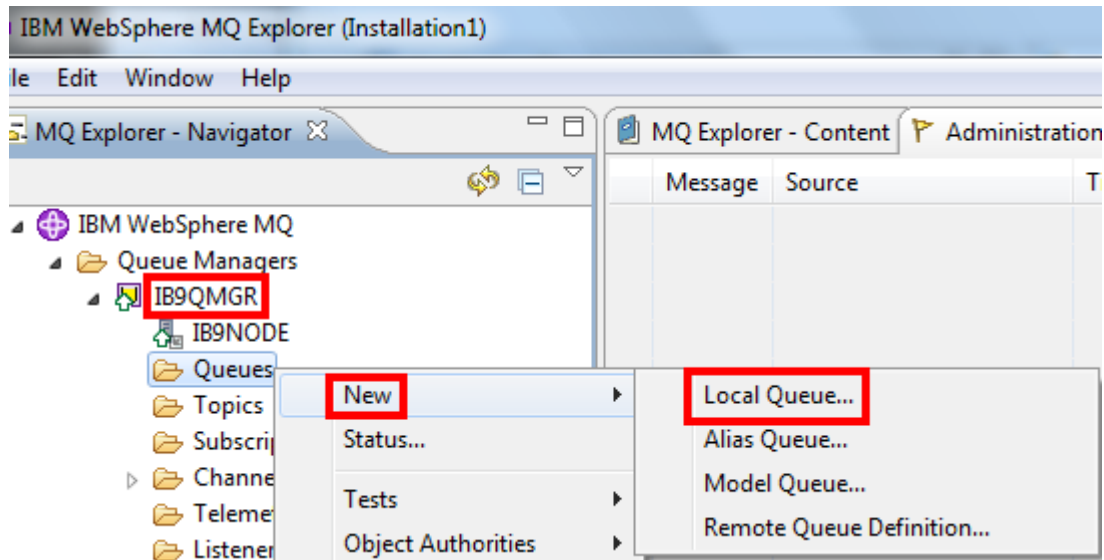
__29. Press the **Finish** button to create the queue.



__30. Press the **OK** button to dismiss the dialog box.

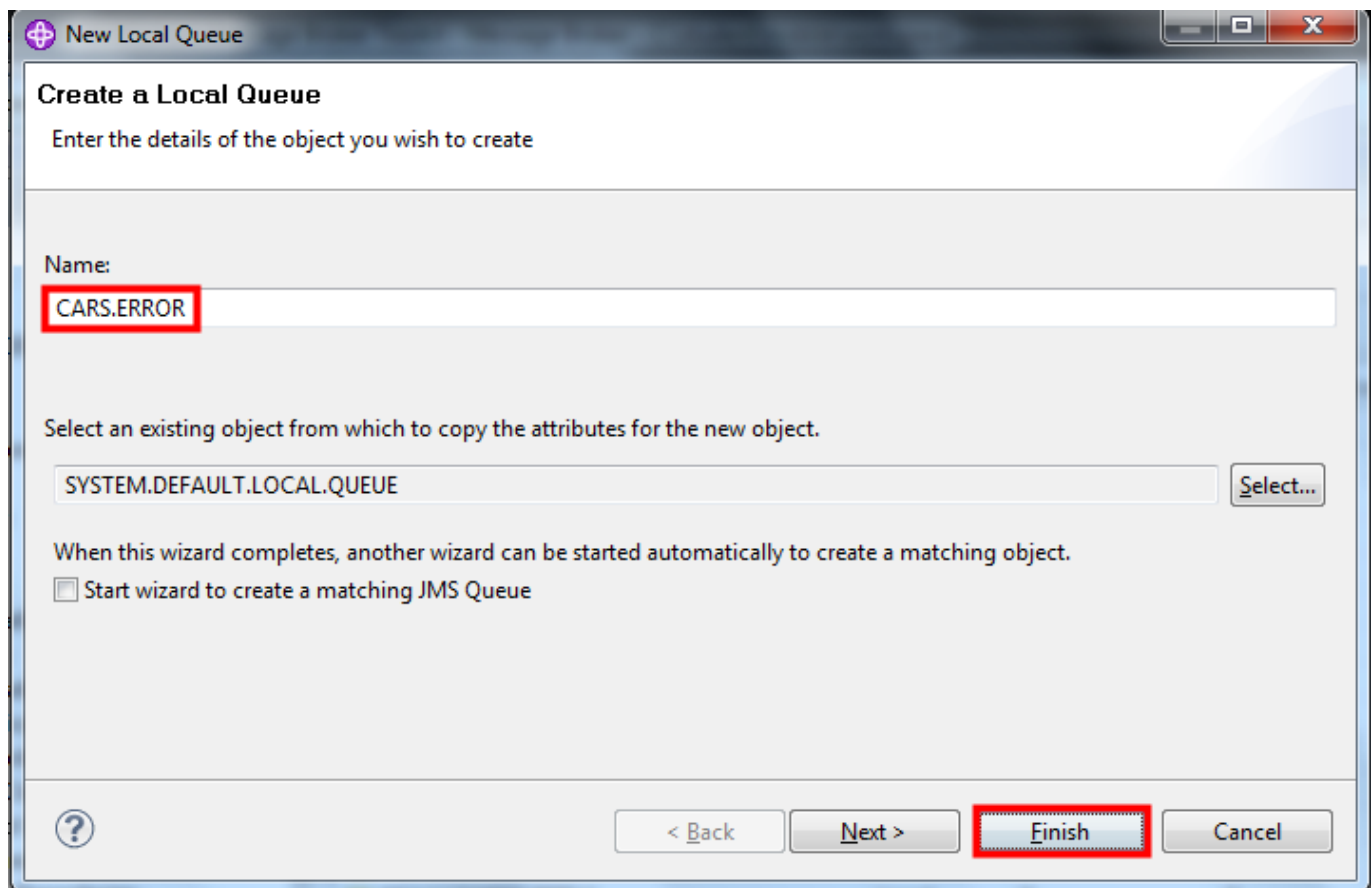


- __31. Select the **Queues** folder.
- __32. Press the right mouse button.
- __33. Select **New→Local Queue** from the menu.



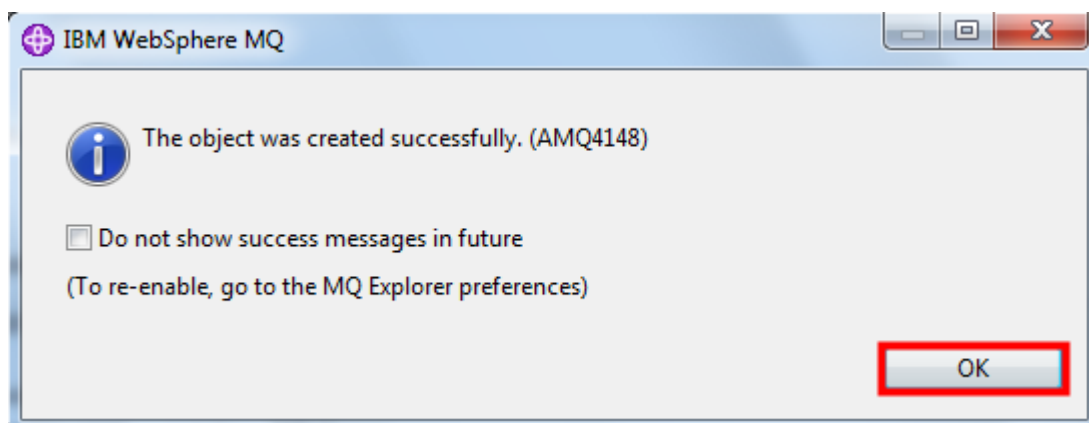
__34. Enter **CARS.ERROR** as the **Name** of the queue.

__35. Press the **Finish** button to create the queue.



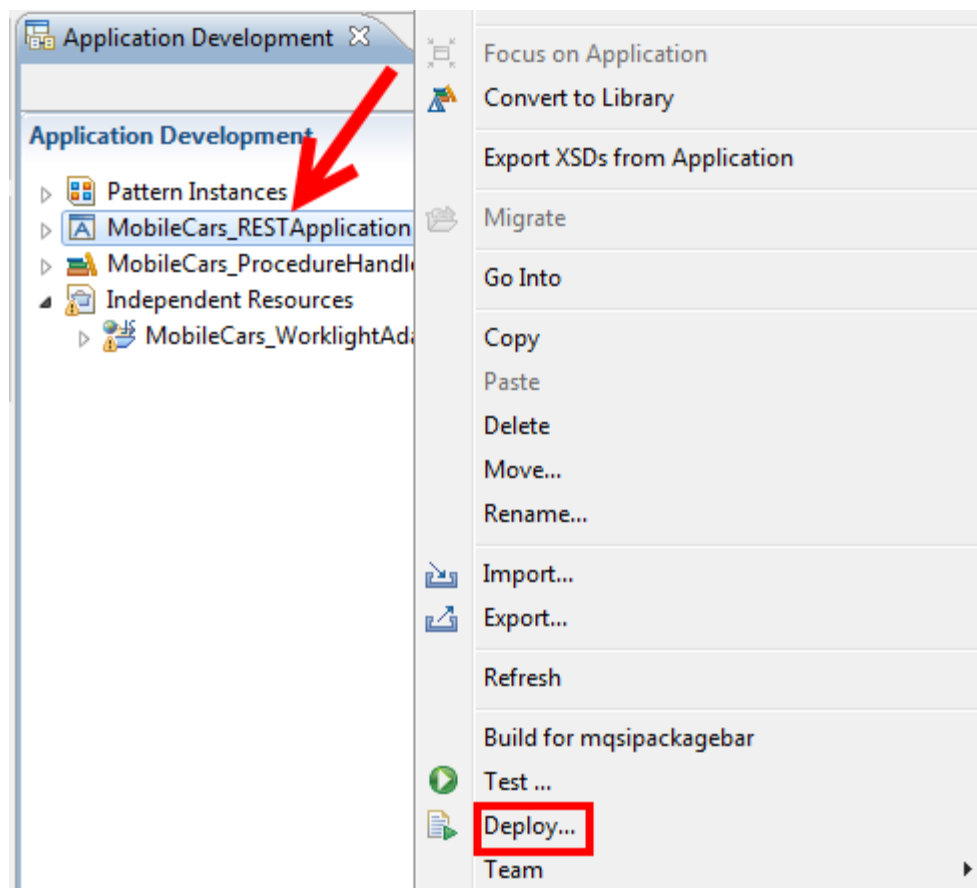
The image shows a 'New Local Queue' dialog box with the title 'Create a Local Queue'. Below the title is the instruction 'Enter the details of the object you wish to create'. The 'Name:' field contains 'CARS.ERROR', which is highlighted with a red rectangle. Below this is a section 'Select an existing object from which to copy the attributes for the new object.' with a text box containing 'SYSTEM.DEFAULT.LOCAL.QUEUE' and a 'Select...' button. Further down, there is a checkbox labeled 'Start wizard to create a matching JMS Queue' which is currently unchecked. At the bottom, there are four buttons: '< Back', 'Next >', 'Finish' (highlighted with a red rectangle), and 'Cancel'.

__36. Press the **OK** button to dismiss the dialog box.

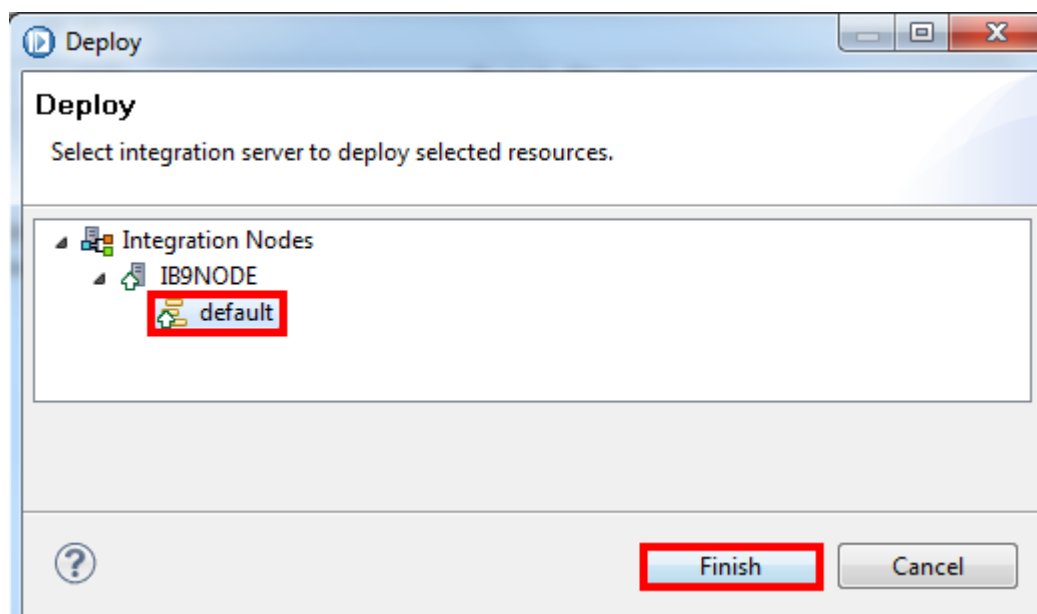


The image shows an 'IBM WebSphere MQ' dialog box with an information icon and the text 'The object was created successfully. (AMQ4148)'. Below this is a checkbox labeled 'Do not show success messages in future' which is unchecked, followed by the text '(To re-enable, go to the MQ Explorer preferences)'. At the bottom right, there is an 'OK' button highlighted with a red rectangle.

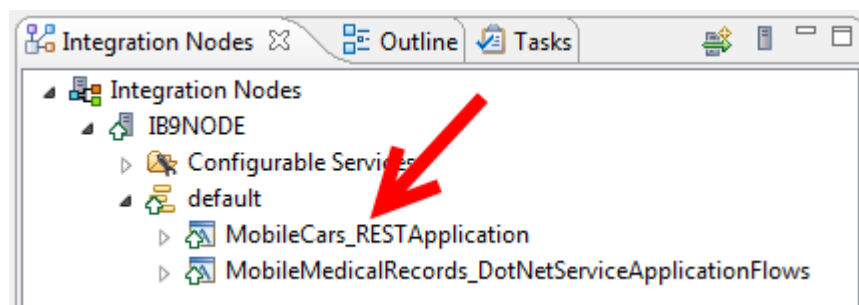
- __37. Minimize MQ Explorer.
- __38. Select the **MobileCars_RestApplication** application.
- __39. Press the right mouse button.
- __40. Select **Deploy** from the menu.



- __41. Select the **default** integration server.
- __42. Press the **Finish** button to initiate the deployment.



The deployed application should be visible in the **Integration Nodes** view.

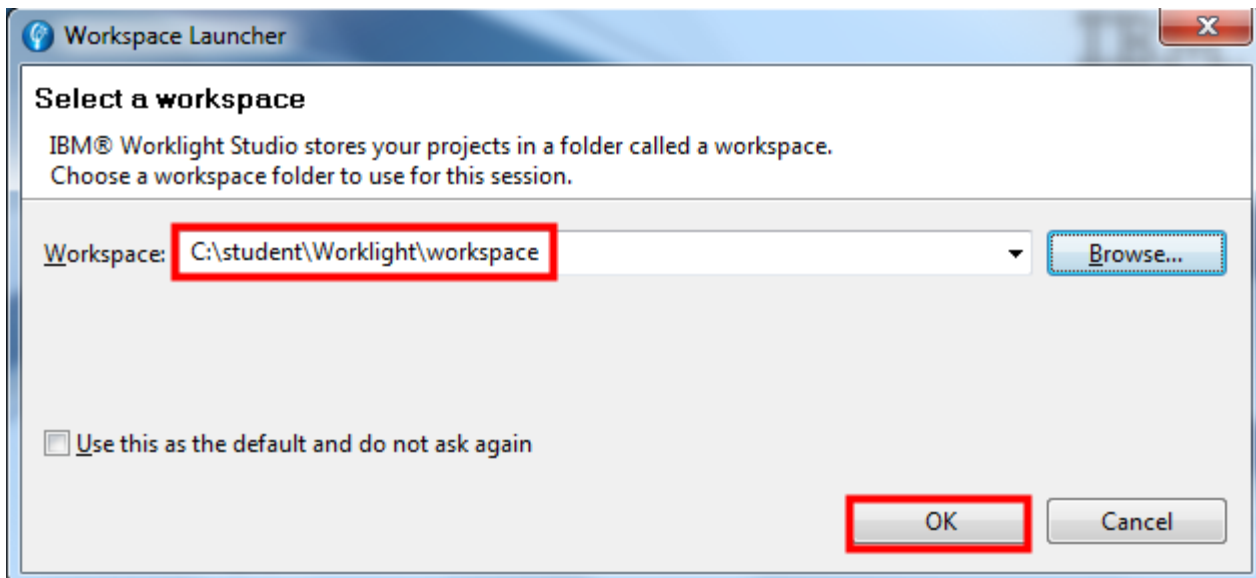


6.6 Install and test the MobileCars adapter

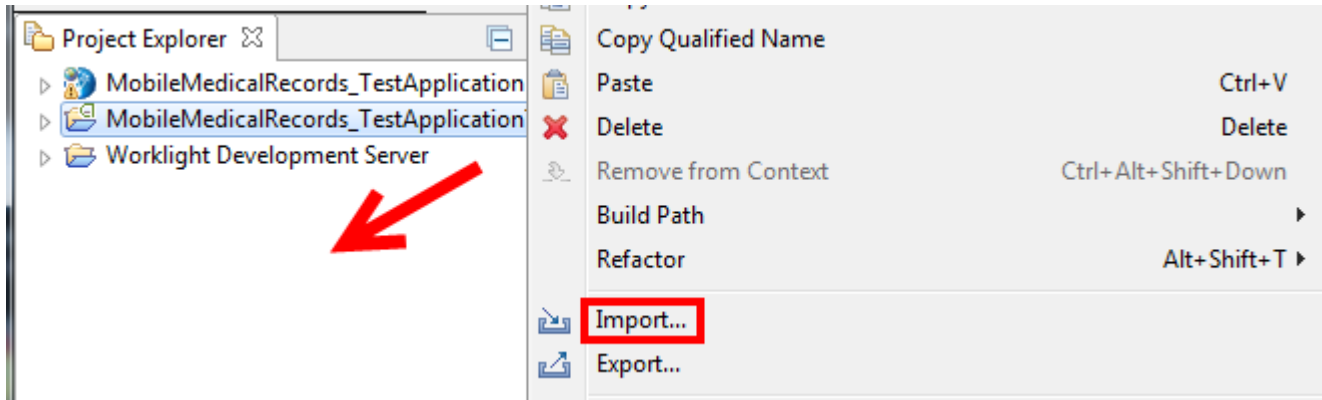
1. Either return to or open the Eclipse Workbench for Worklight, using the icon on the main window.



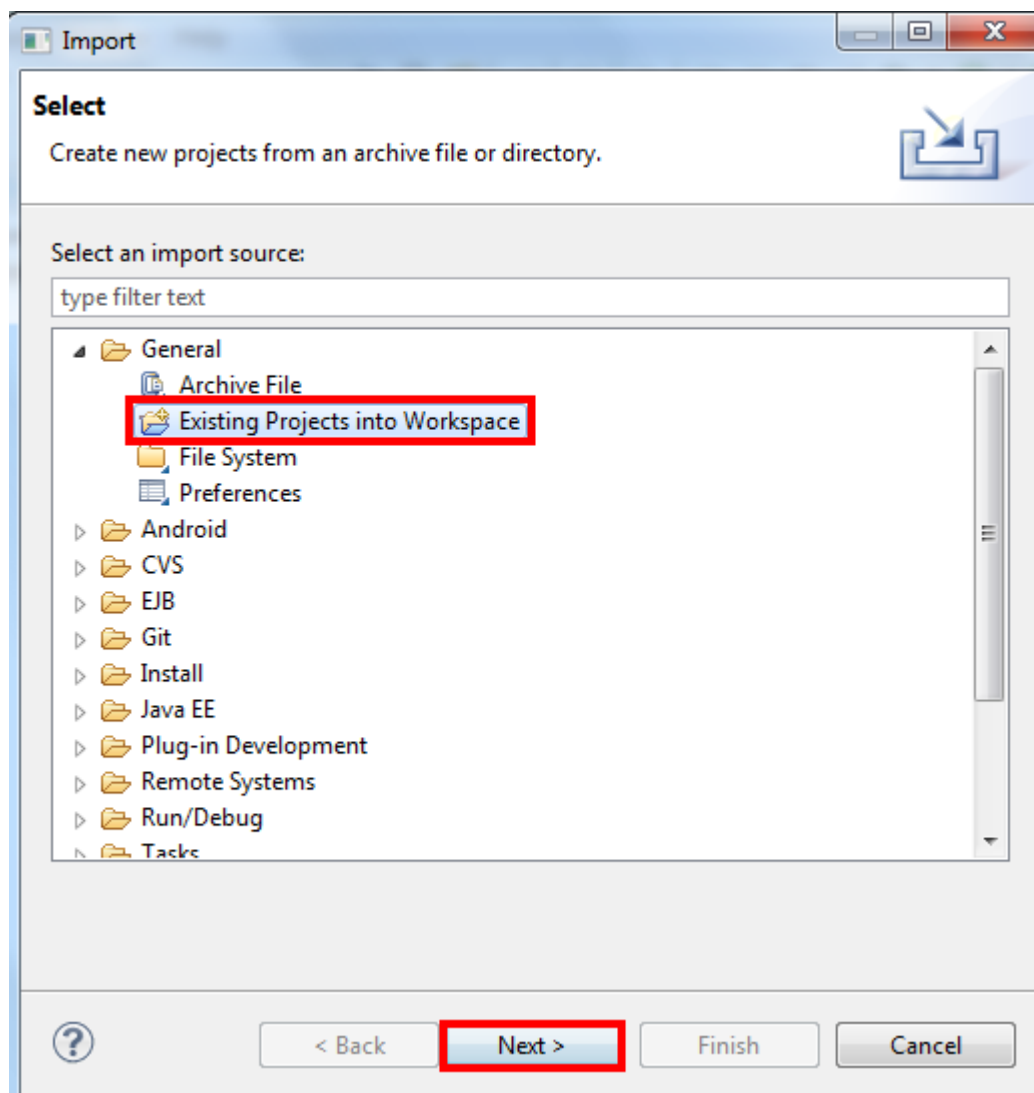
2. Select the **C:\student\Worklight\workspace** directory.
3. Press the **OK** button to open the workbench.



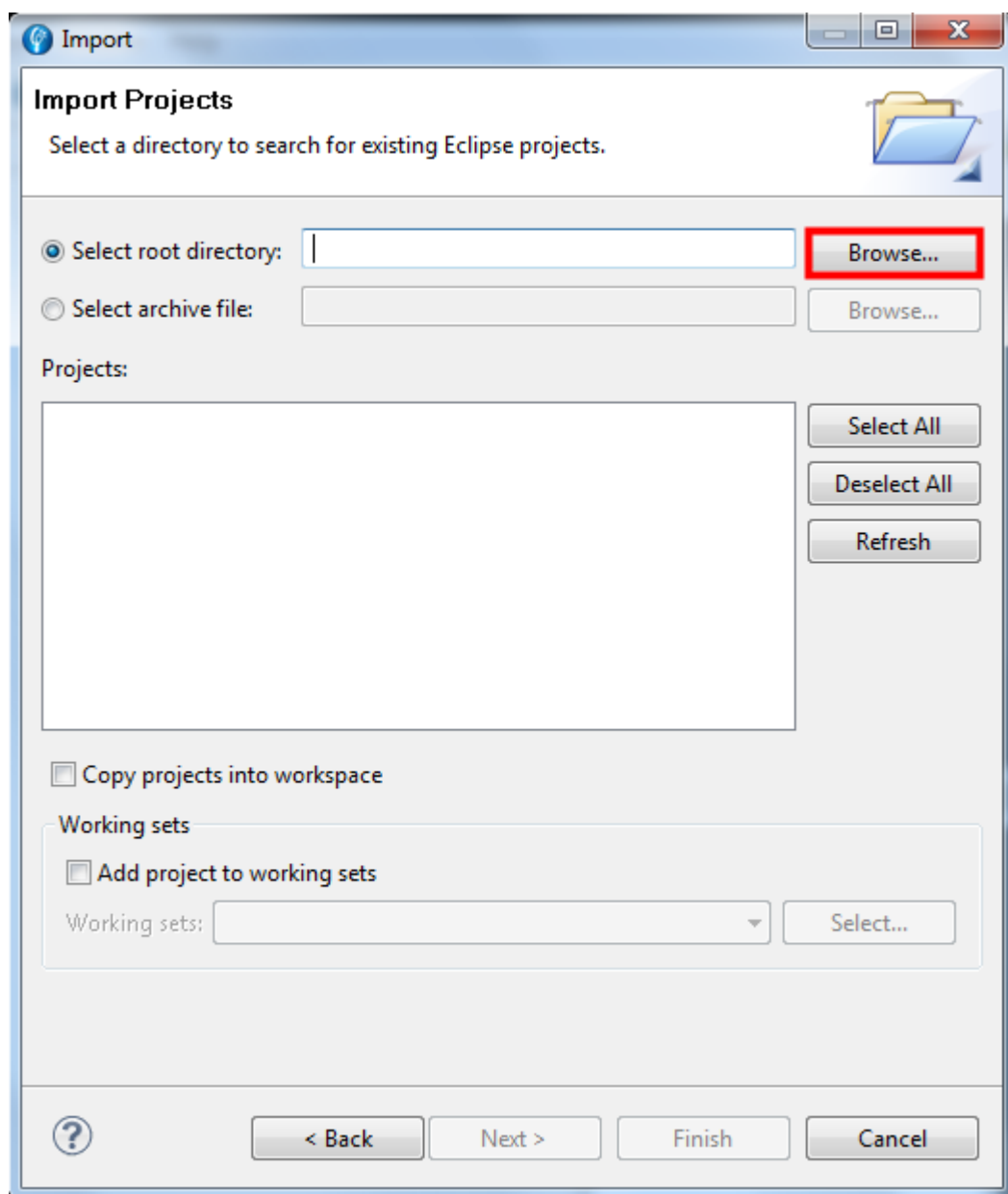
- ___4. In the **Project Explorer** point to a blank space.
- ___5. Press the right mouse button.
- ___6. Select **Import** from the menu.



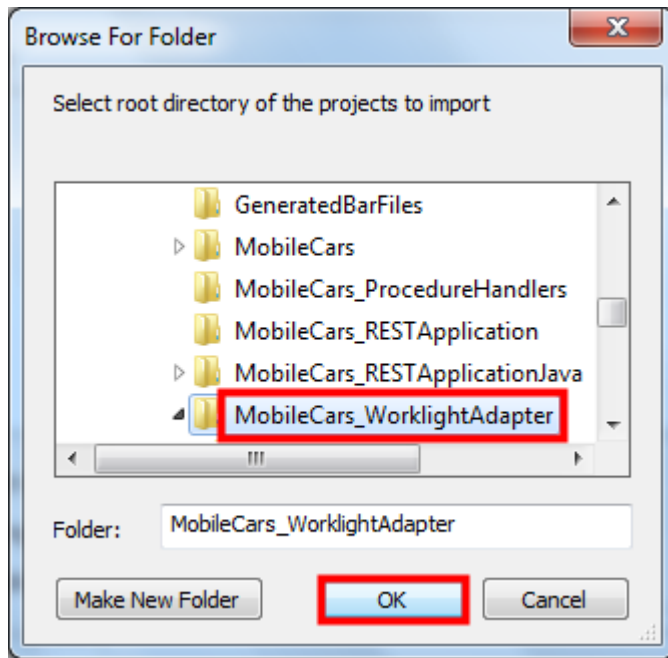
- __7. Expand the **General** folder.
- __8. Select **Existing Projects into Workspace**.
- __9. Press the **Next** button.



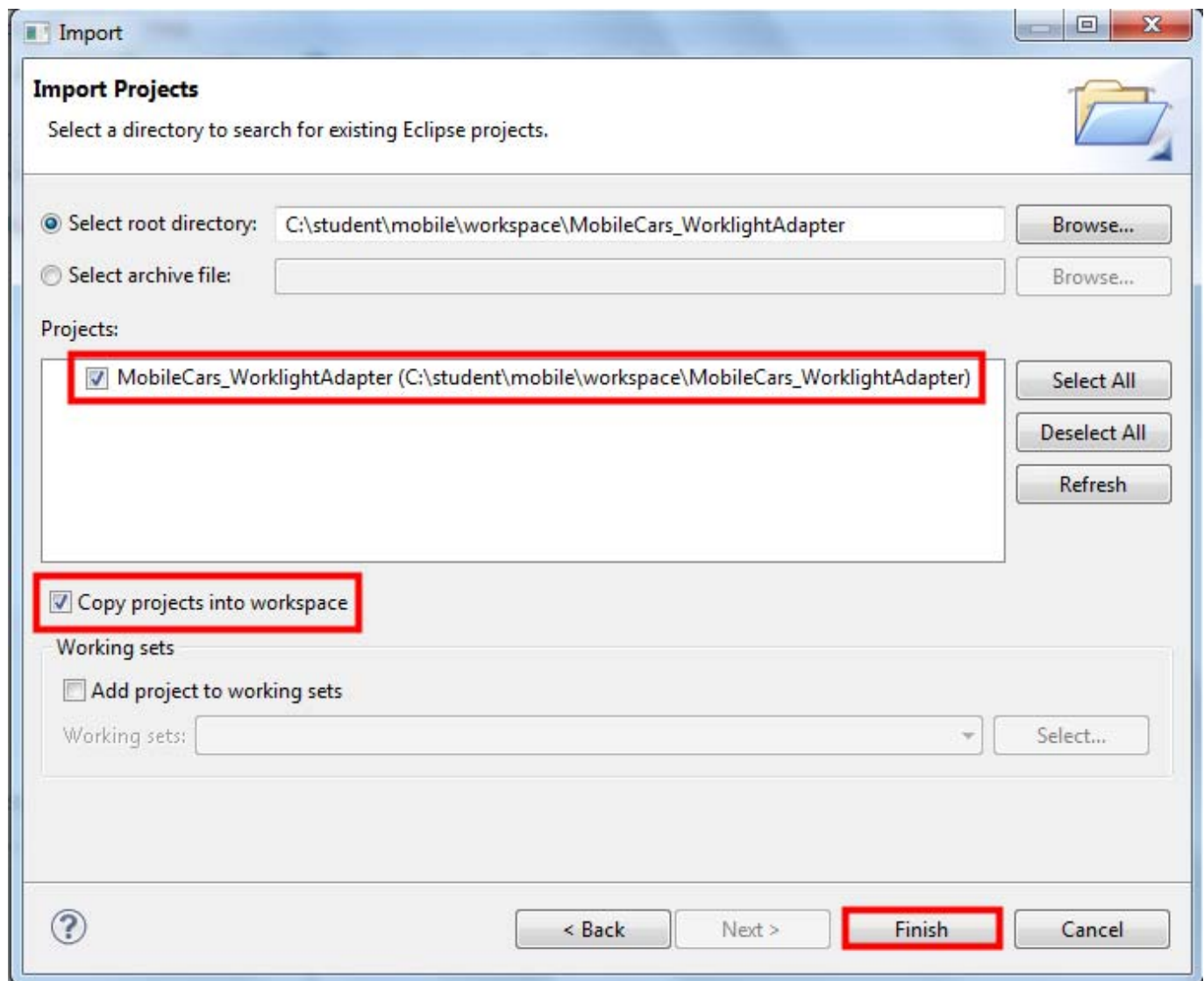
___10. Press the **Browse** button next to the **Select root directory** text box.



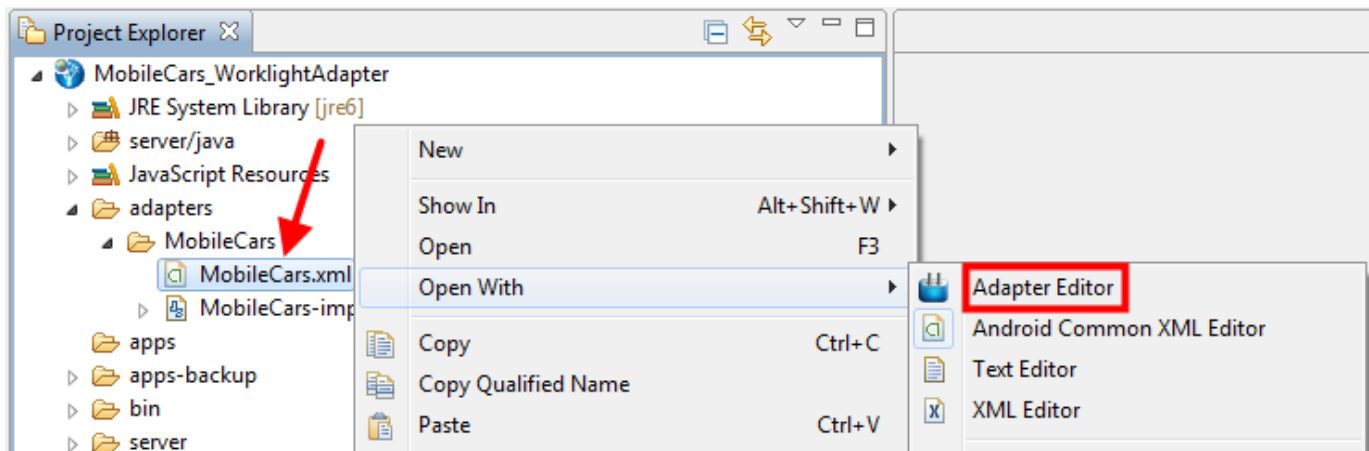
- __11. Navigate to the **C:\student\mobile\workspace\MobileCars_WorklightAdapter** project.
- __12. Press the **OK** button.



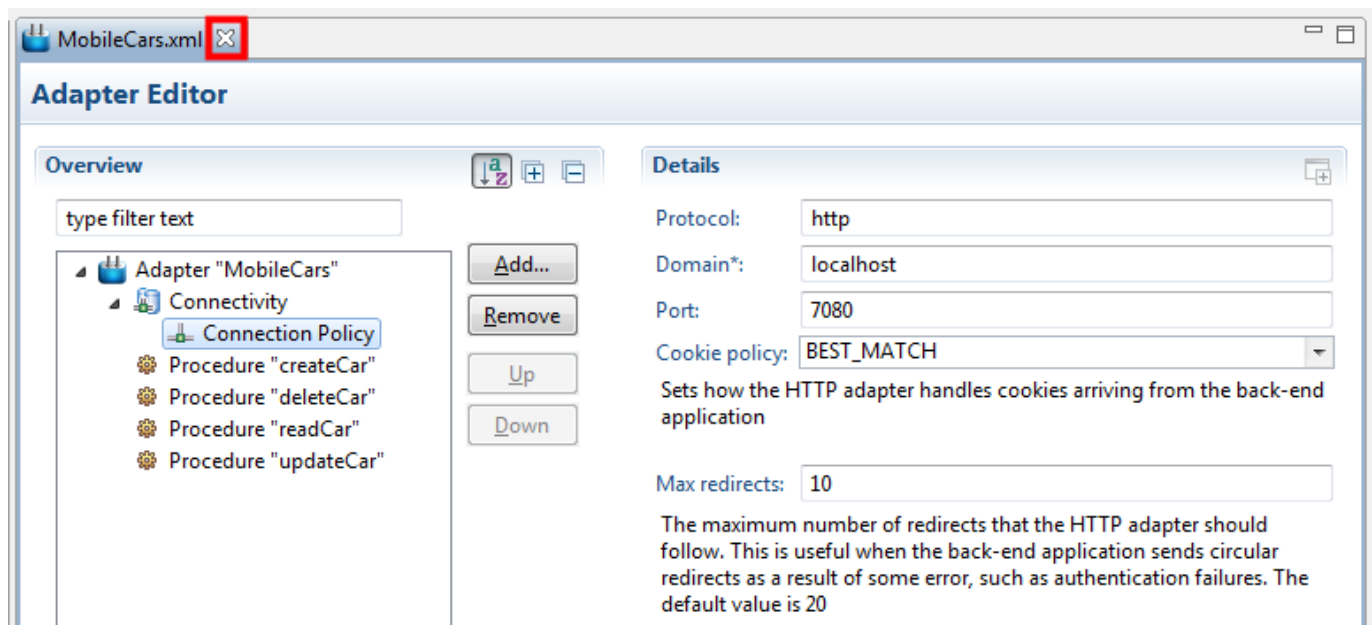
- __13. Select the **Copy projects into workspace** check box.
- __14. Press the **Finish** button to start the import.



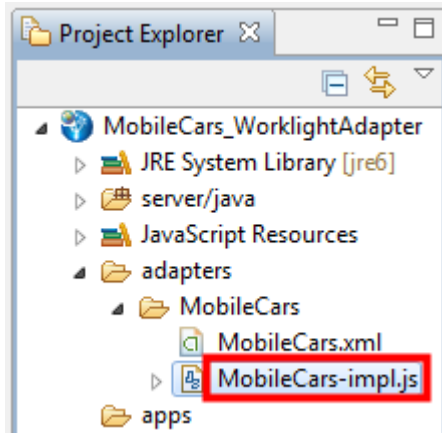
- __15. Expand **MobileCars_WorklightAdapter**→**adapters**→**MobileCars**.
- __16. Select the **MobileCars.xml** file.
- __17. Press the right mouse button.
- __18. Select **Open With**→**Adapter Editor** from the menu.



- __19. Examine the contents of the adapter XML file.
- __20. When finished close the editor.



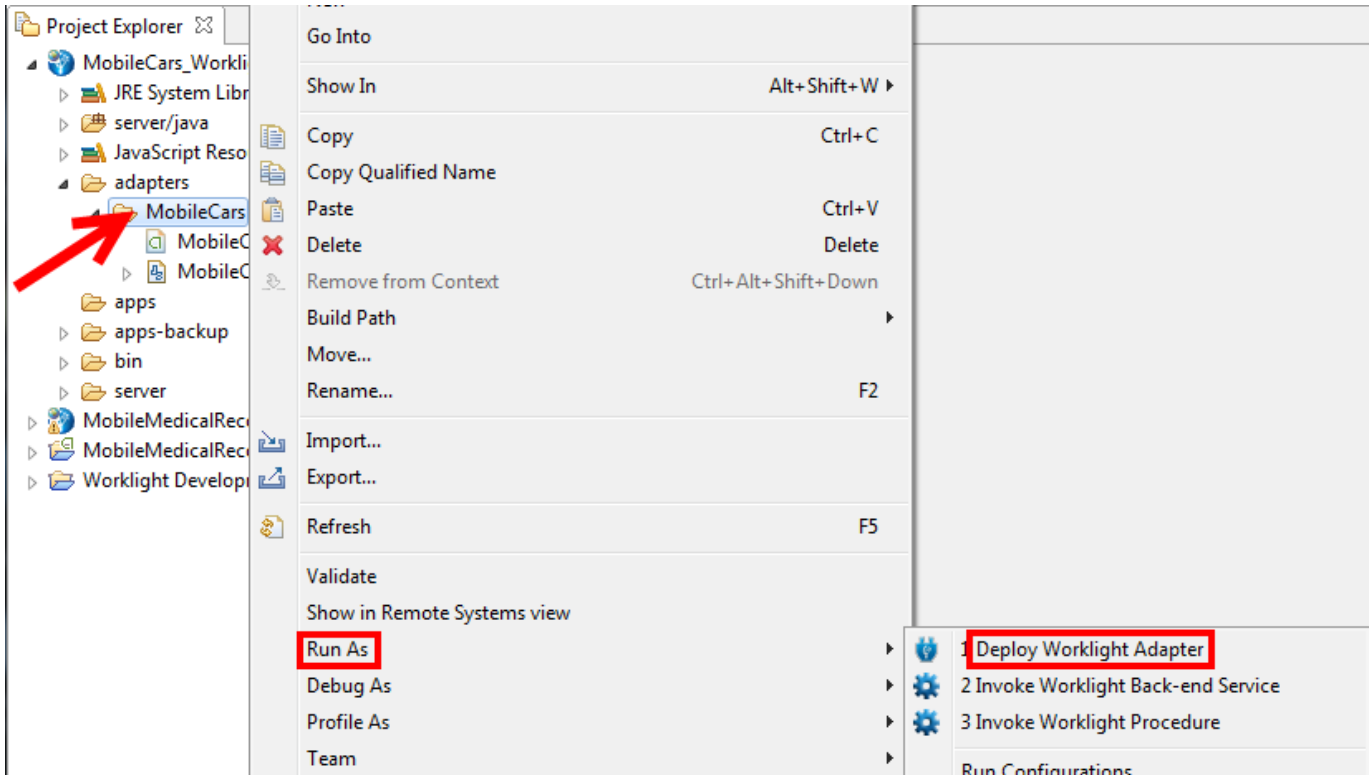
__21. Double click the **MobileCars-impl.js** JavaScript file.



__22. After examining the JavaScript file close the editor pane.

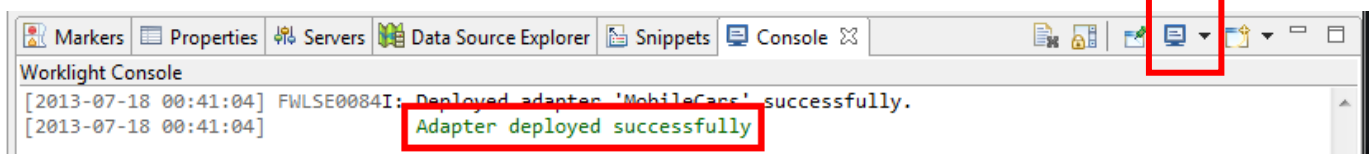


- __23. Select the **MobileCars** adapter.
- __24. Press the right mouse button.
- __25. Select **Run As→Deploy Worklight Adapter** from the menu.



The results of the adapter deployment should be visible in the console view.

- __26. Confirm that the adapter deployment was successful. It may be necessary to select the **Worklight Console** as the selected console, using the icon highlighted below.



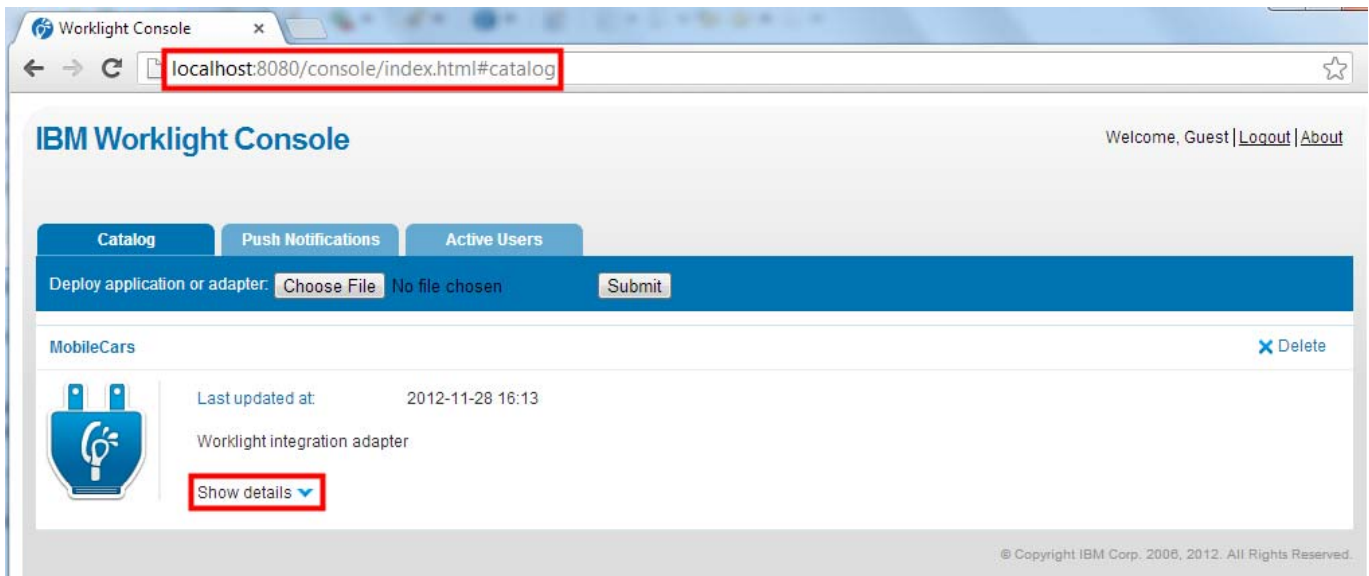
Start a Google Chrome browser session.

- __27. Double-click the Chrome icon.



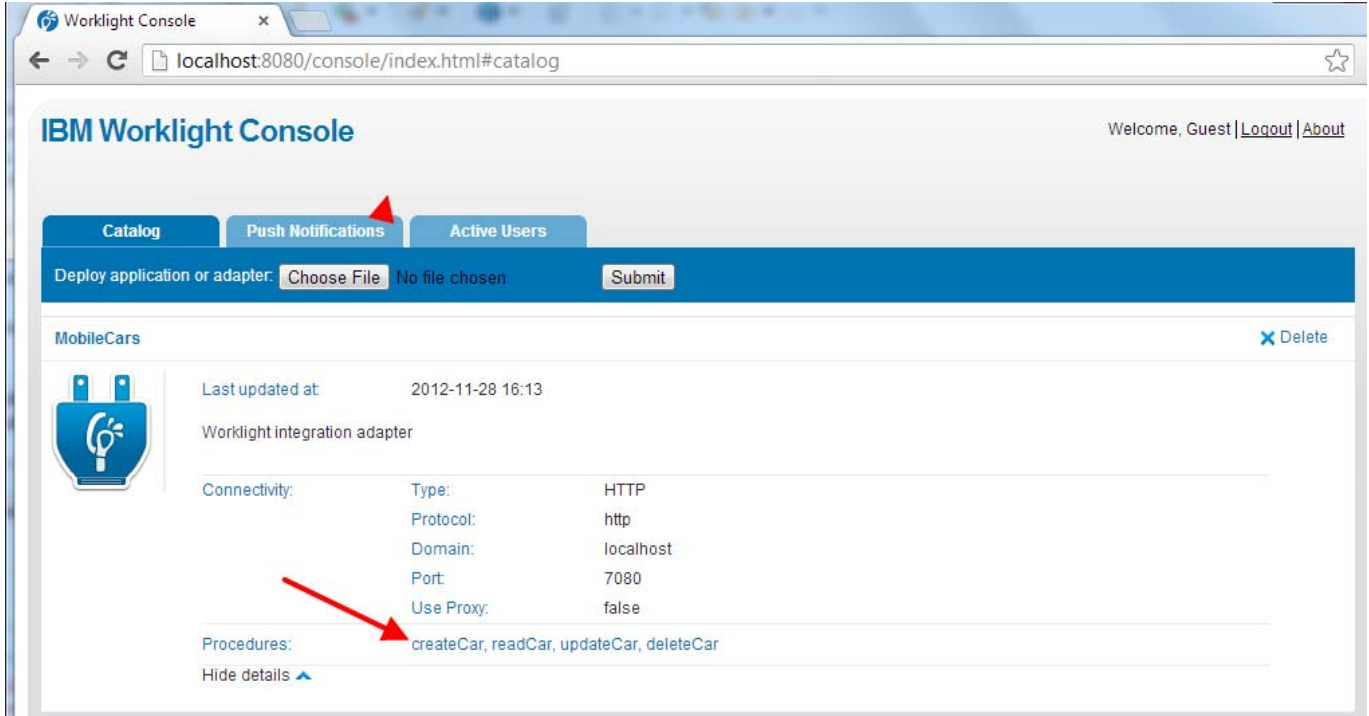
__28. Navigate to <http://localhost:8080/console>.

__29. Expand the **Show details** area.

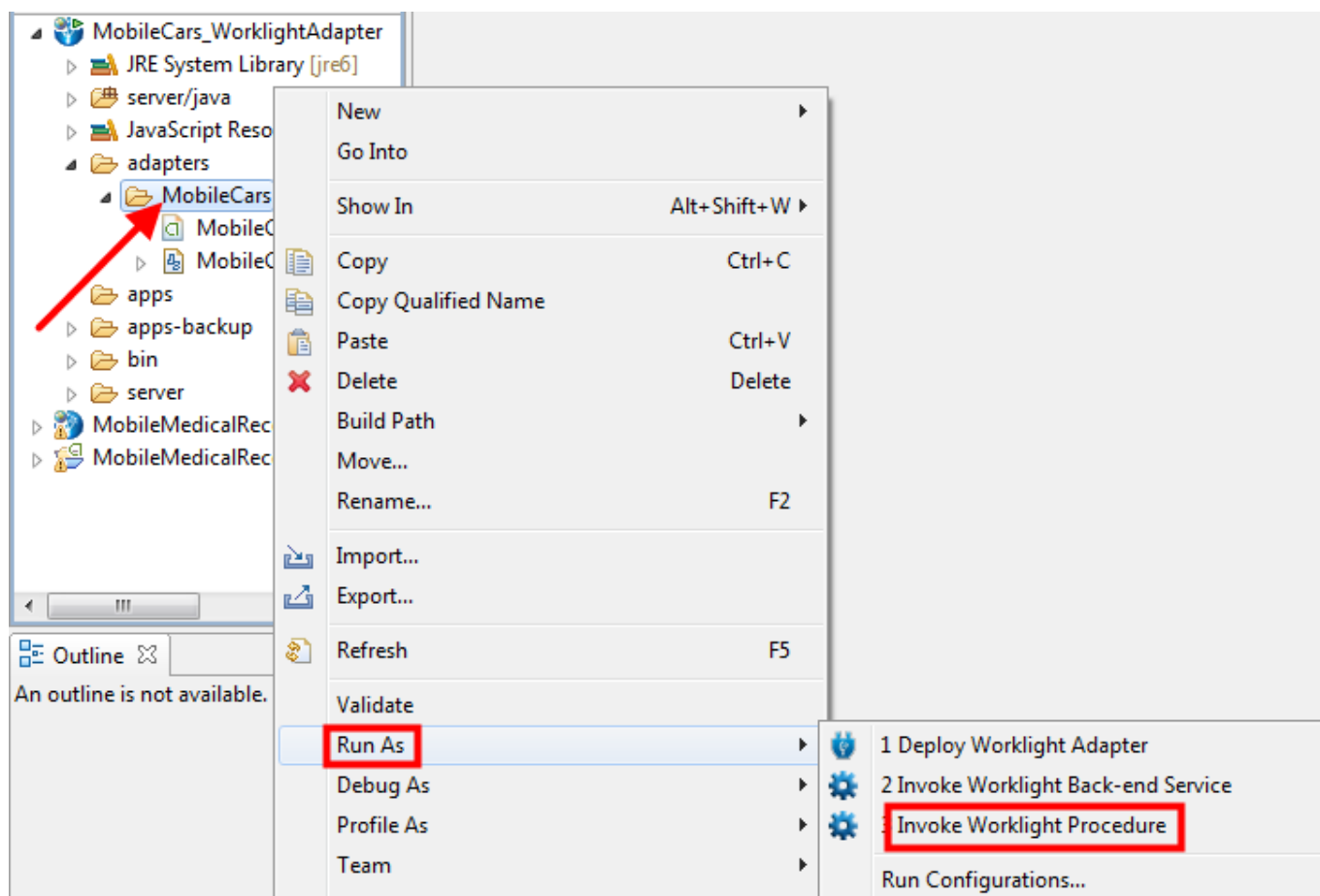


The procedures in the installed adapter should be visible.

__30. Minimize the browser session.



- __31. Return to the Worklight studio
- __32. Select the **MobileCars** adapter.
- __33. Press the right mouse button.
- __34. Select **Run As**→**Invoke Worklight Procedure** from the menu.



The Worklight test client should open.

__35. Use the drop-down menu to select **createCar** as the **Procedure name**.

Edit Configuration

Edit configuration and launch.

Name: Invoke Procedure MobileCars_WorklightAdapter - MobileCars

Invoke Procedure Data

Project name: MobileCars_WorklightAdapter

Adapter name: MobileCars

Procedure name: **createCar**

Signature:
createCar (parameters)

Parameters (comma-separated):

Apply Revert

Run Close

The data for this procedure is available in the **C:\student\mobile\data** directory in a file named **volvo.txt**. The contents of the file can be copied and pasted into the **Parameters** field.

__36. Enter **"{\Car\:{\Make\:\Volvo\,\Colour\:\Red\}}"** in the **Parameters** field (N.B. string values are enclosed in double quotes so the double quotes are part of the input).

__37. Press the **Run** Button.

Edit Configuration

Edit configuration and launch.

Name: Invoke Procedure MobileCars_WorklightAdapter - MobileCars

Invoke Procedure Data

Project name: MobileCars_WorklightAdapter

Adapter name: MobileCars

Procedure name: createCar

Signature:
createCar (parameters)

Parameters (comma-separated):
"{\Car\:{\Make\:\Volvo\,\Colour\:\Red\}}"

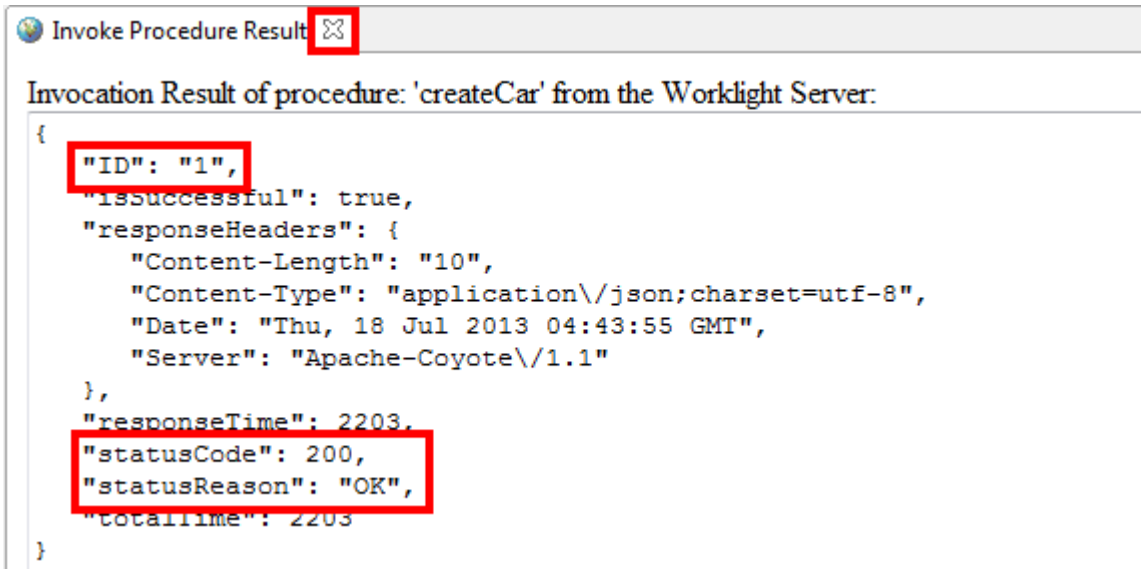
Apply Revert

? Run Close

The Worklight test client will invoke the MobileCars Worklight adapter, which in turn will invoke the message flow.

The Test Client will show the value that has been returned from the Integration Bus application when the **createCar** procedure is invoked. In the example below, this shows a result of **1** for the **ID** field.

__38. After examining the results close the **Invoke Procedure Result** window.

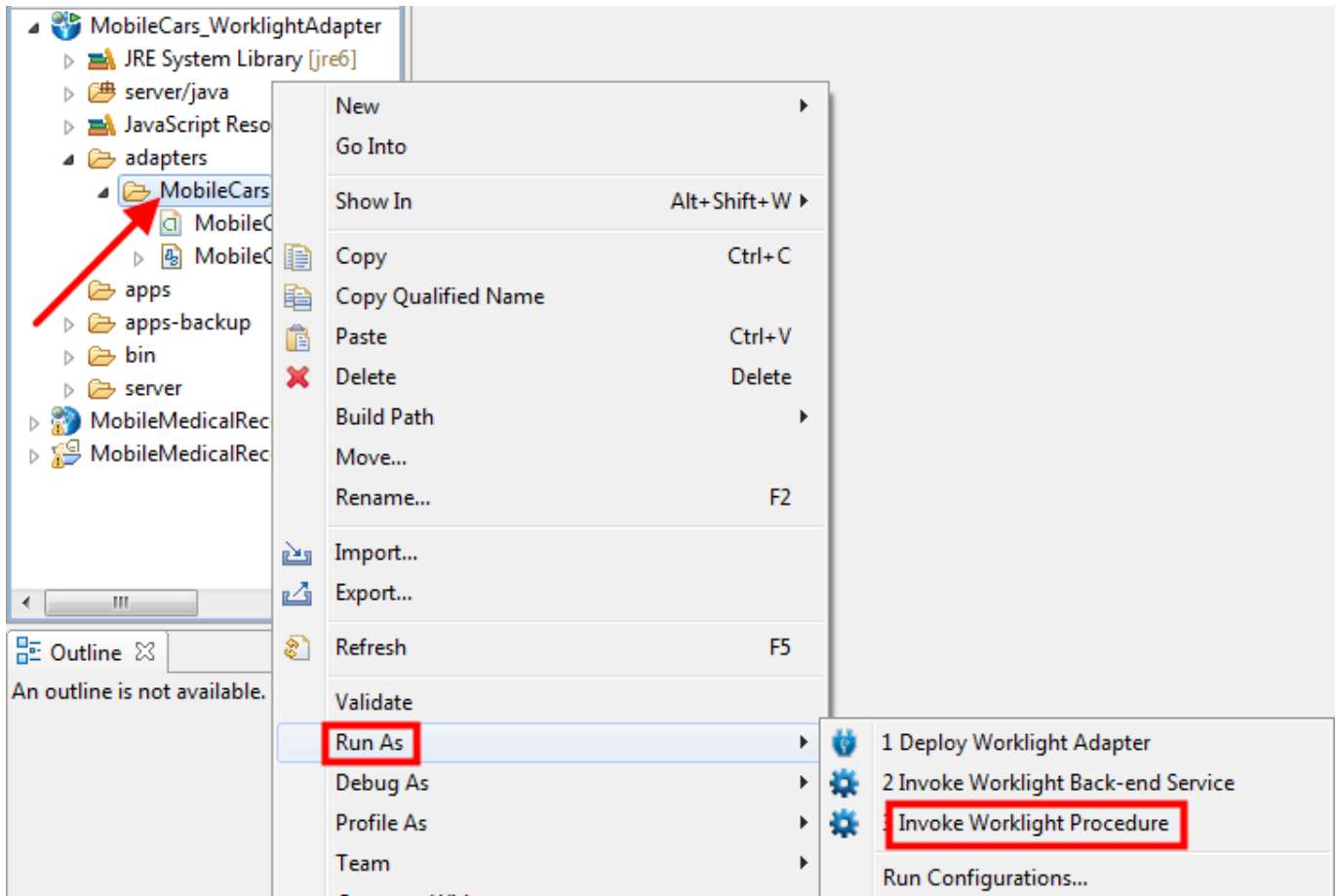


A second and third car will be added next.

__39. Select the **MobileCars** adapter.

__40. Press the right mouse button.

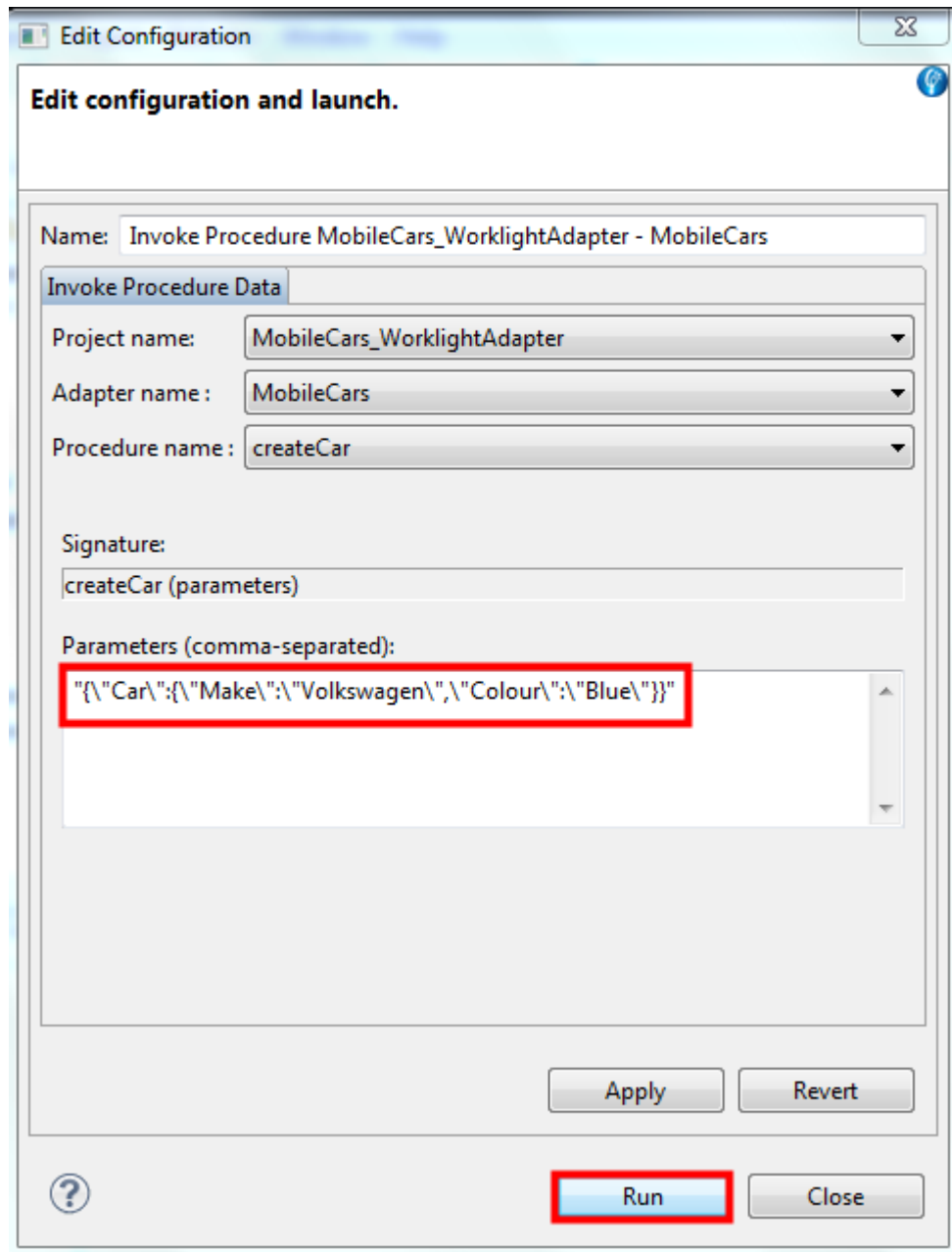
__41. Select **Run As→Invoke Worklight Procedure** from the menu.



The data for this procedure is available in the **C:\student\mobile\data** directory in a file named **volkswagen.txt**. The contents of the file can be copied and pasted into the **Parameters** field.

__42. Enter `{"Car\":{\"Make\":\"Volkswagen\",\"Colour\":\"Blue\"}}` in the **Parameters** field (N.B. string values are enclosed in double quotes so the double quotes are part of the input).

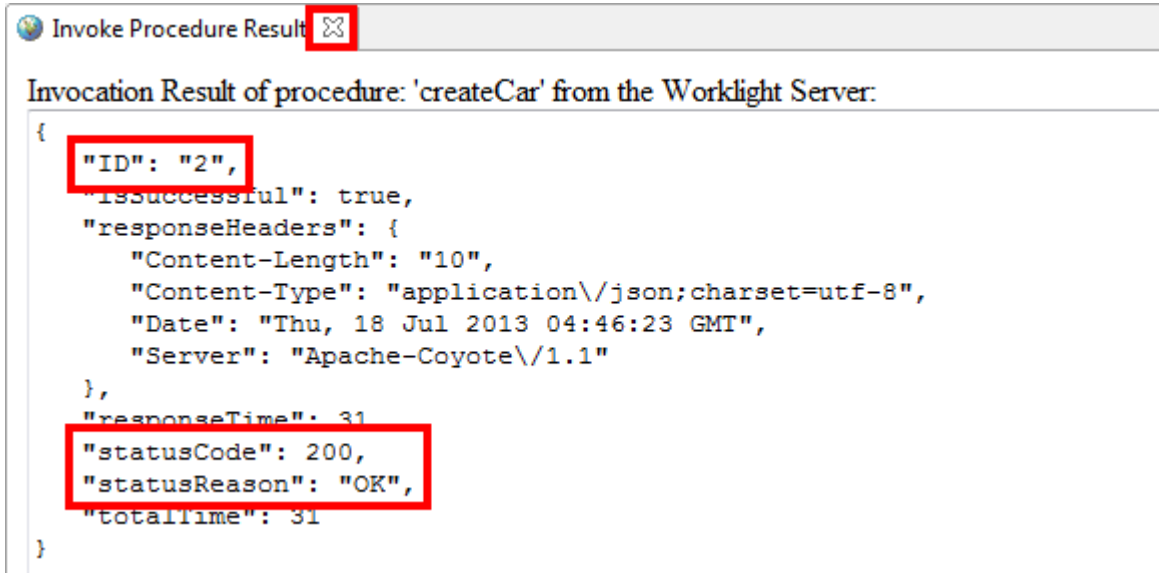
__43. Press the **Run** Button.




The Worklight test client will invoke the MobileCars Worklight adapter, which in turn will invoke the message flow.

The Test Client will show the value that has been returned from the Integration Bus application when the **createCar** procedure is invoked. In the example below, this shows a result of **2** for the **ID** field.

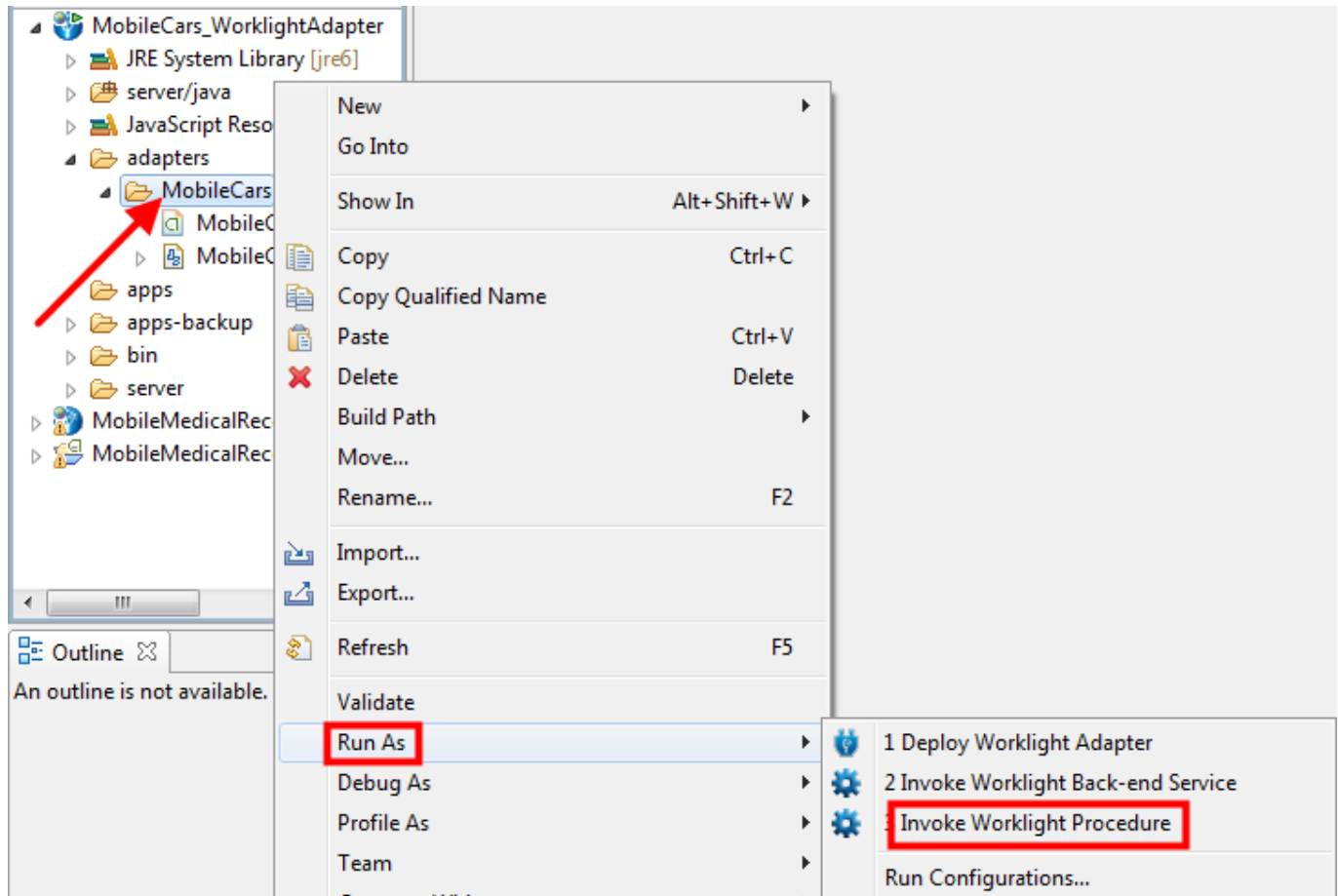
__44. After examining the results close the **Invoke Procedure Result** window.



```
Invoke Procedure Result 
Invocation Result of procedure: 'createCar' from the Worklight Server:
{
  "ID": "2",
  "isSuccessful": true,
  "responseHeaders": {
    "Content-Length": "10",
    "Content-Type": "application/json;charset=utf-8",
    "Date": "Thu, 18 Jul 2013 04:46:23 GMT",
    "Server": "Apache-Coyote/1.1"
  },
  "responseTime": 31,
  "statusCode": 200,
  "statusReason": "OK",
  "totalTime": 31
}
```

A third car will now be added.

- __45. Select the **MobileCars** adapter.
- __46. Press the right mouse button.
- __47. Select **Run As**→**Invoke Worklight Procedure** from the menu.



The data for this procedure is available in the **C:\student\mobile\data** directory in a file named **vauxhall.txt**. The contents of the file can be copied and pasted into the **Parameters** field.

__48. Enter **"{"Car":{"Make":"Vauxhall"},"Colour":"Green"}"** in the **Parameters** field (N.B. string values are enclosed in double quotes so the double quotes are part of the input).

__49. Press the **Run** Button.

Edit Configuration

Edit configuration and launch.

Name: Invoke Procedure MobileCars_WorklightAdapter - MobileCars

Invoke Procedure Data

Project name: MobileCars_WorklightAdapter

Adapter name: MobileCars

Procedure name: createCar

Signature:
createCar (parameters)

Parameters (comma-separated):
{"Car":{"Make":"Vauxhall"},"Colour":"Green"}

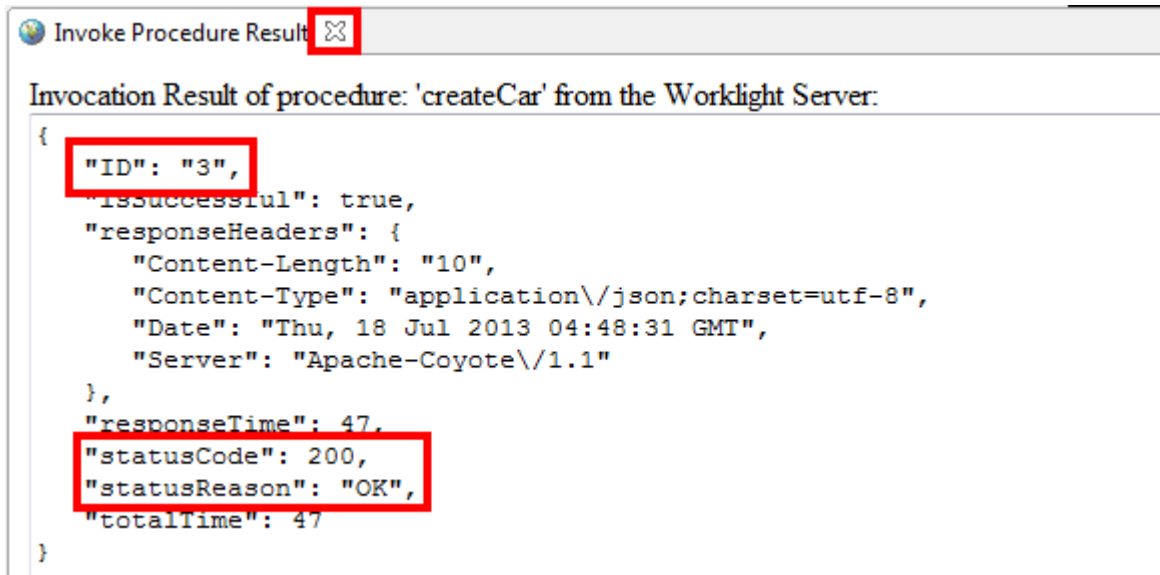
Apply Revert

? Run Close

The Worklight test client will invoke the MobileCars Worklight adapter, which in turn will invoke the message flow.

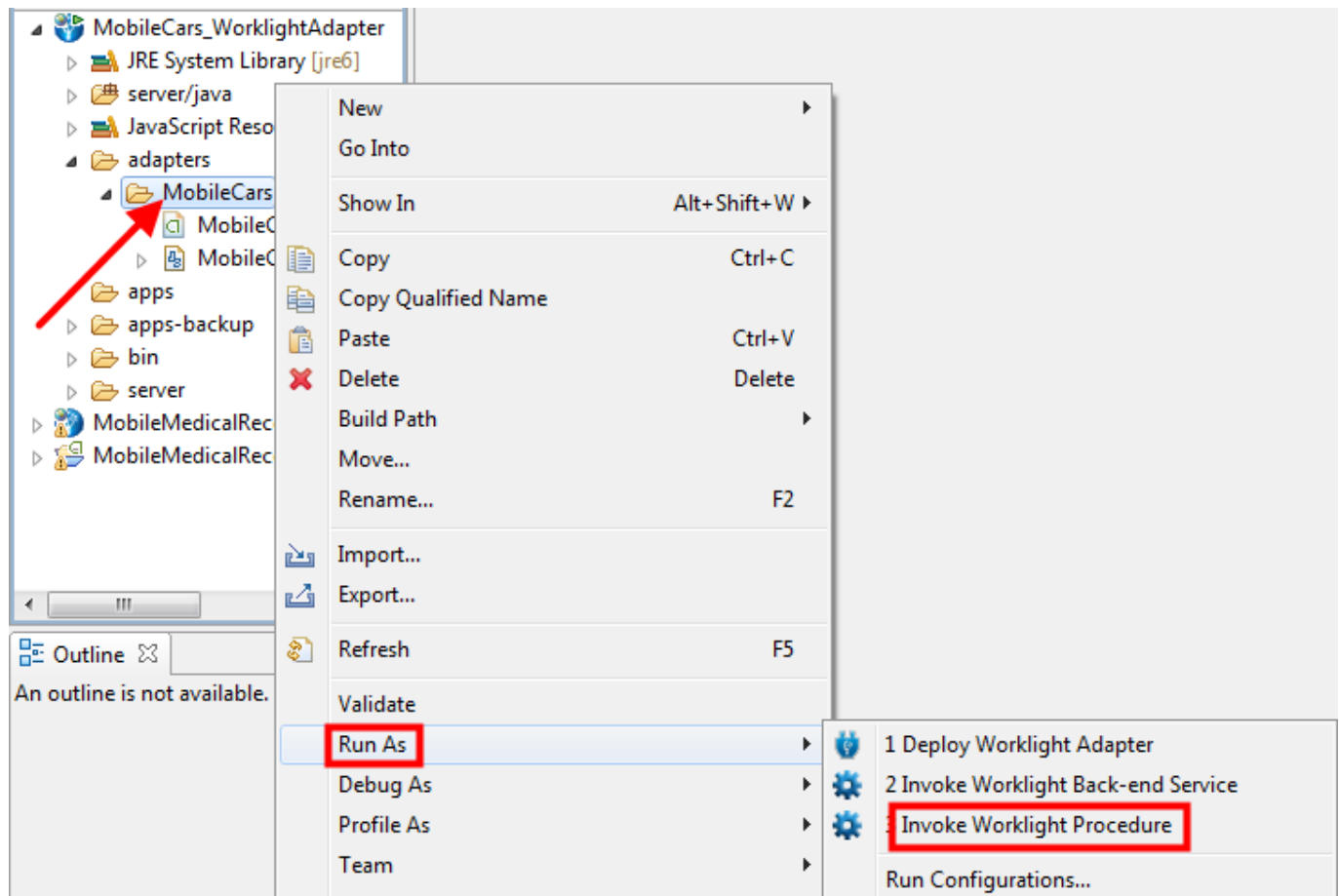
The Test Client will show the value that has been returned from the Integration Bus application when the **createCar** procedure is invoked. In the example below, this shows a result of **3** for the **ID** field.

__50. After examining the results close the **Invoke Procedure Result** window.

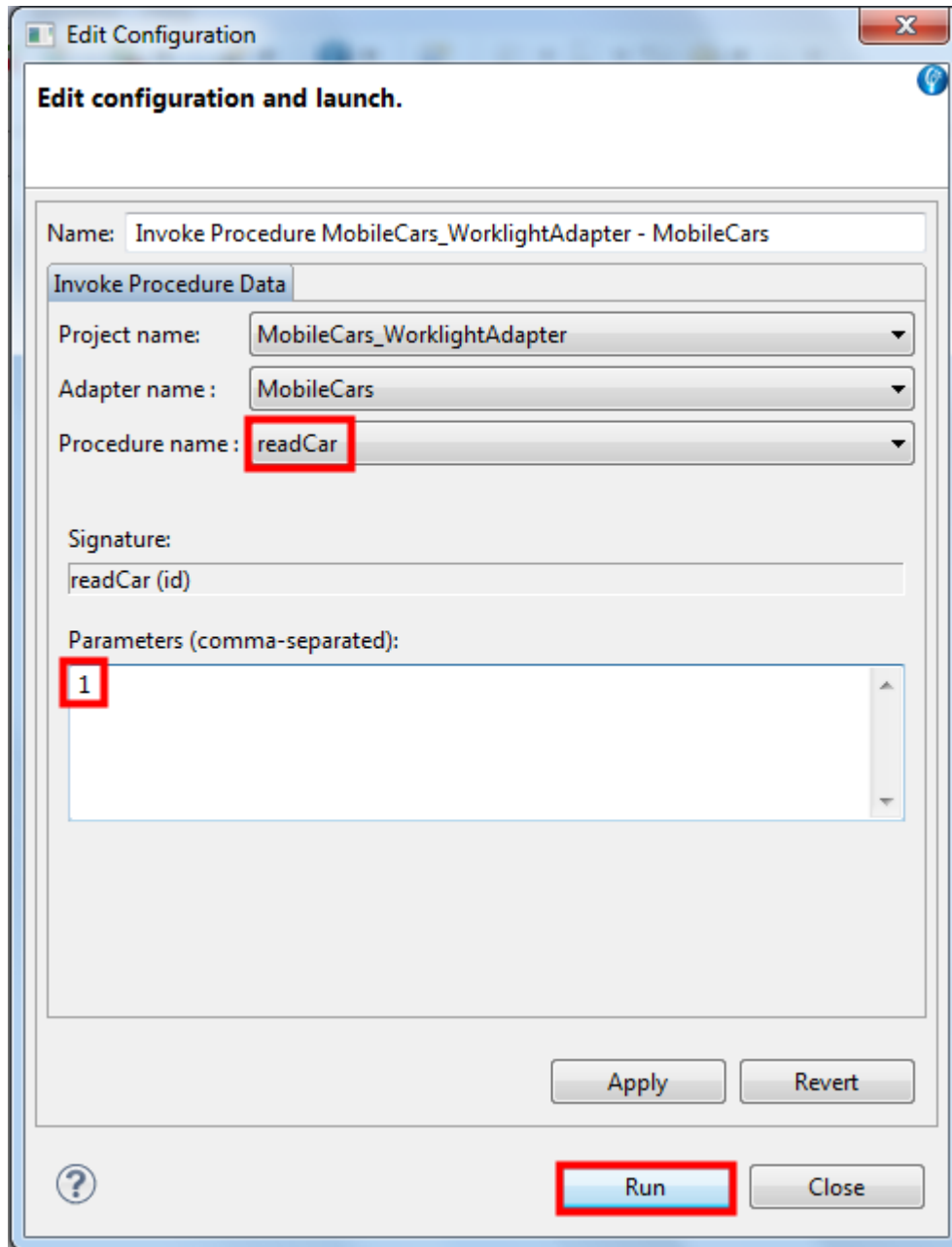


The next part of the lab will retrieve two of the car objects that were created in the previous steps.

- __51. Select the **MobileCars** adapter.
- __52. Press the right mouse button.
- __53. Select **Run As→Invoke Worklight Procedure** from the menu.



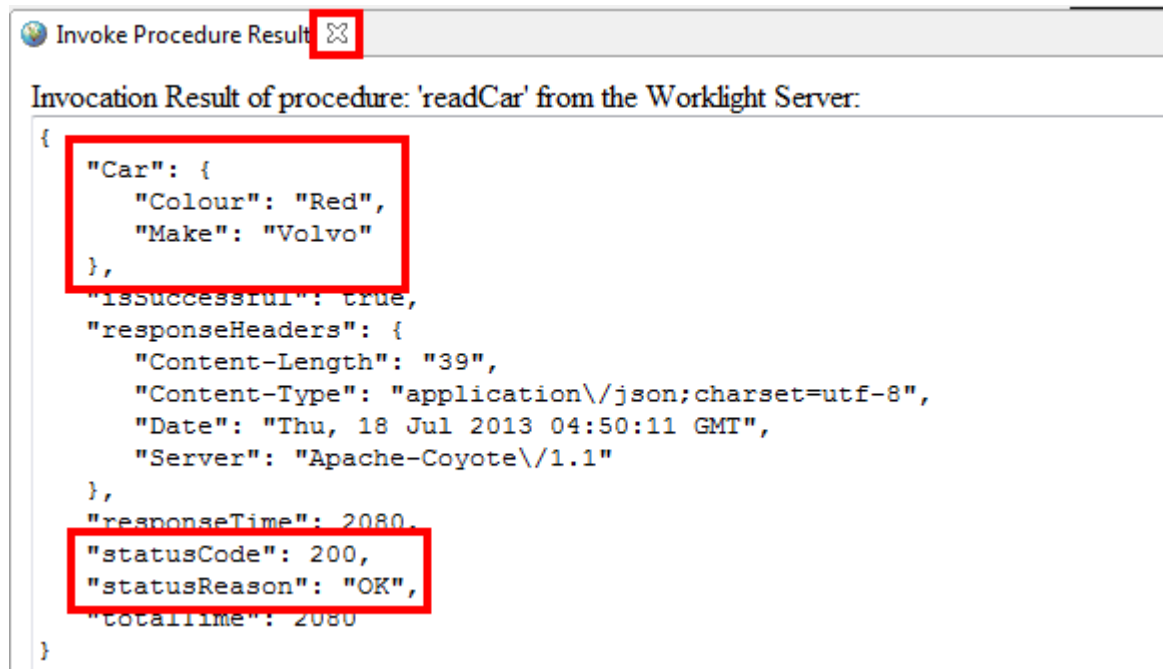
- __54. Use the drop-down menu to select **readCar** as the **Procedure name**.
- __55. Enter **1** in the **Parameters** field.
- __56. Press the **Run** Button.



The Worklight test client will invoke the MobileCars Worklight adapter, which in turn will invoke the message flow.

The Test Client will show the value that has been returned from the Integration Bus application when the **readCar** procedure is invoked.

__57. After examining the results close the **Invoke Procedure Result** window.

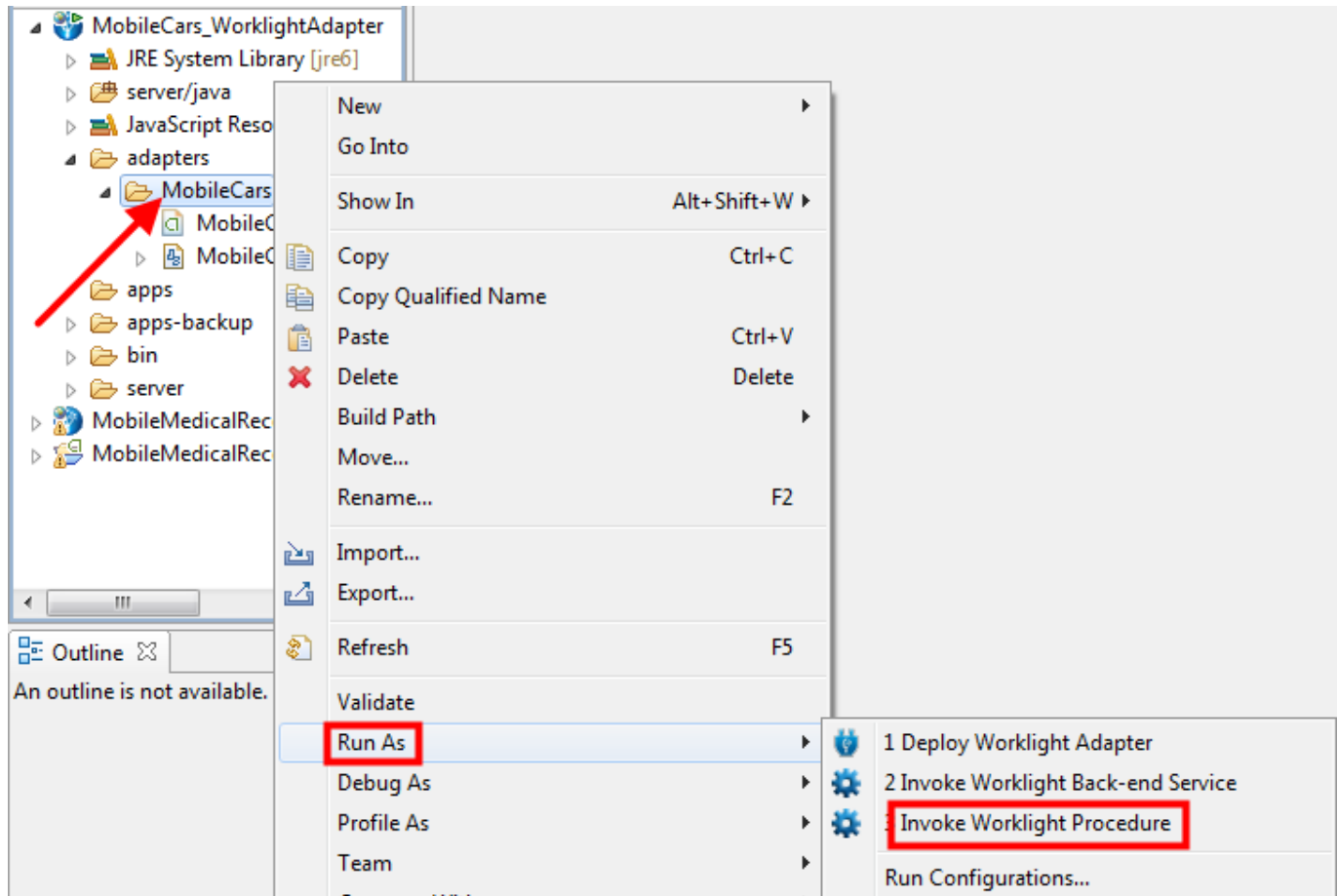


A second car object will be retrieved next.

__58. Select the **MobileCars** adapter.

__59. Press the right mouse button.

__60. Select **Run As**→**Invoke Worklight Procedure** from the menu.



- __61. If necessary, use the drop-down menu to select **readCar** as the **Procedure name**.
- __62. Enter **3** in the **Parameters** field.
- __63. Press the **Run** Button.

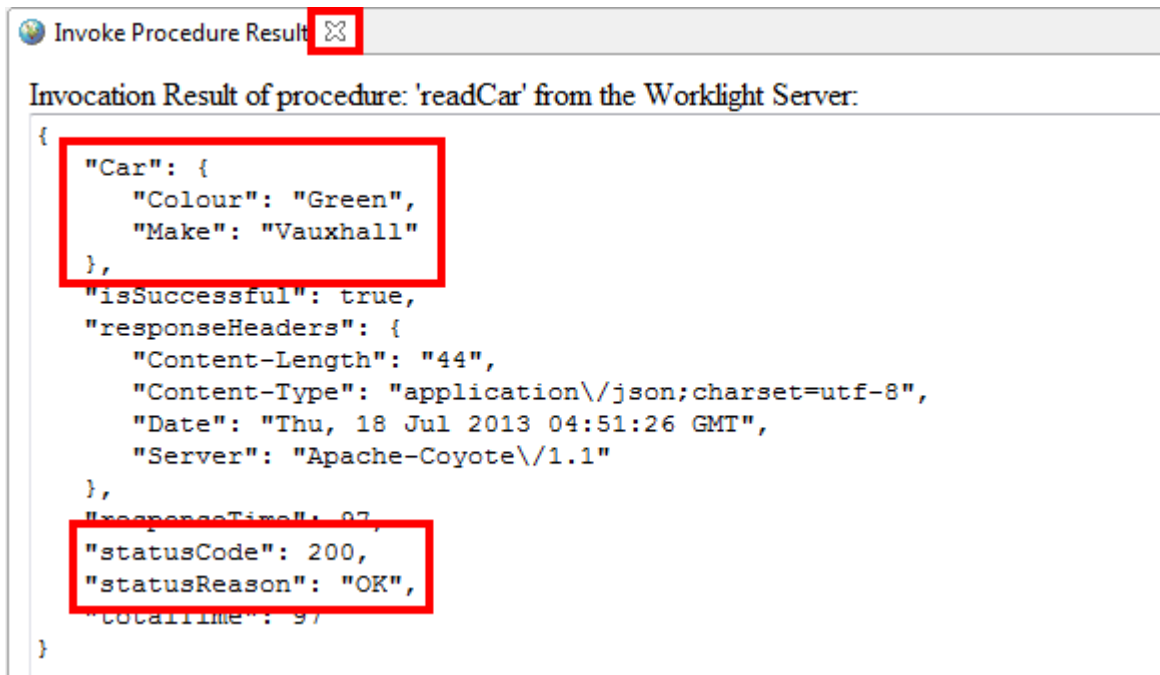
The screenshot shows the 'Edit Configuration' dialog box with the following details:

- Title Bar:** Edit Configuration
- Section Header:** Edit configuration and launch.
- Name:** Invoke Procedure MobileCars_WorklightAdapter - MobileCars
- Invoke Procedure Data Tab:**
 - Project name:** MobileCars_WorklightAdapter
 - Adapter name:** MobileCars
 - Procedure name:** readCar
 - Signature:** readCar (id)
 - Parameters (comma-separated):** 3
- Buttons:** Apply, Revert, Run (highlighted), Close

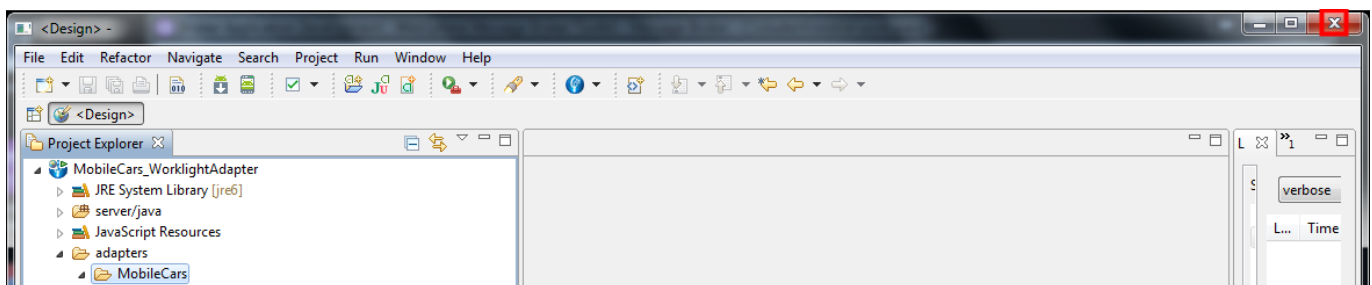
The Worklight test client will invoke the MobileCars Worklight adapter, which in turn will invoke the message flow.

The Test Client will show the value that has been returned from the Integration Bus application when the **readCar** procedure is invoked.

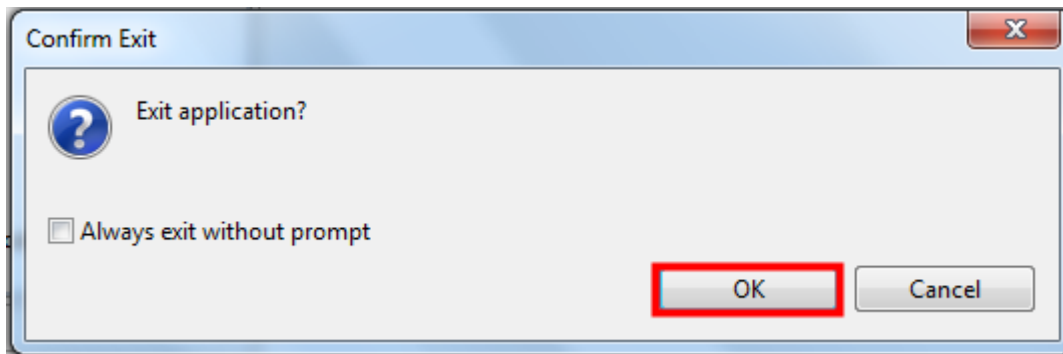
__64. After examining the results close the **Invoke Procedure Result** window.



__65. Close the Worklight studio.

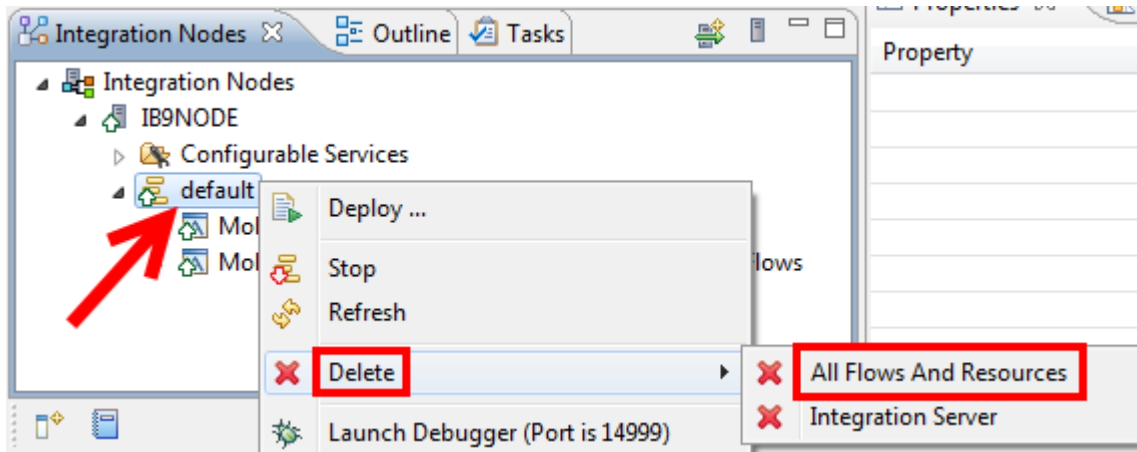


__66. Press the **OK** button to confirm the exit.

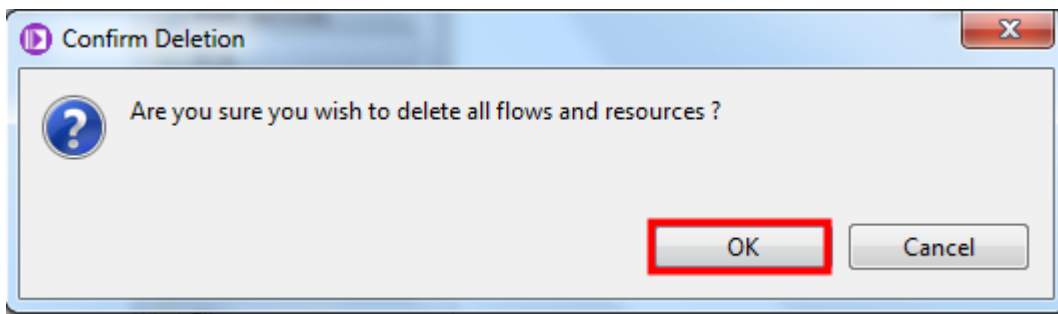


6.7 Clean up

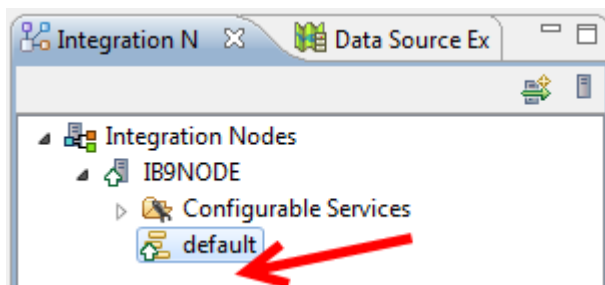
- __1. In the **Integration Nodes** pane of the integration toolkit expand the **IB9NODE** entry.
- __2. Select the **default** integration server.
- __3. Press the right mouse button.
- __4. Select **Delete→All Flows and Resources** from the menu.



- __5. Press the **OK** button to acknowledge the warning and start the deletion.



The **default** integration server should now be empty.



This is the end of lab 6.

Lab 7 Working with files

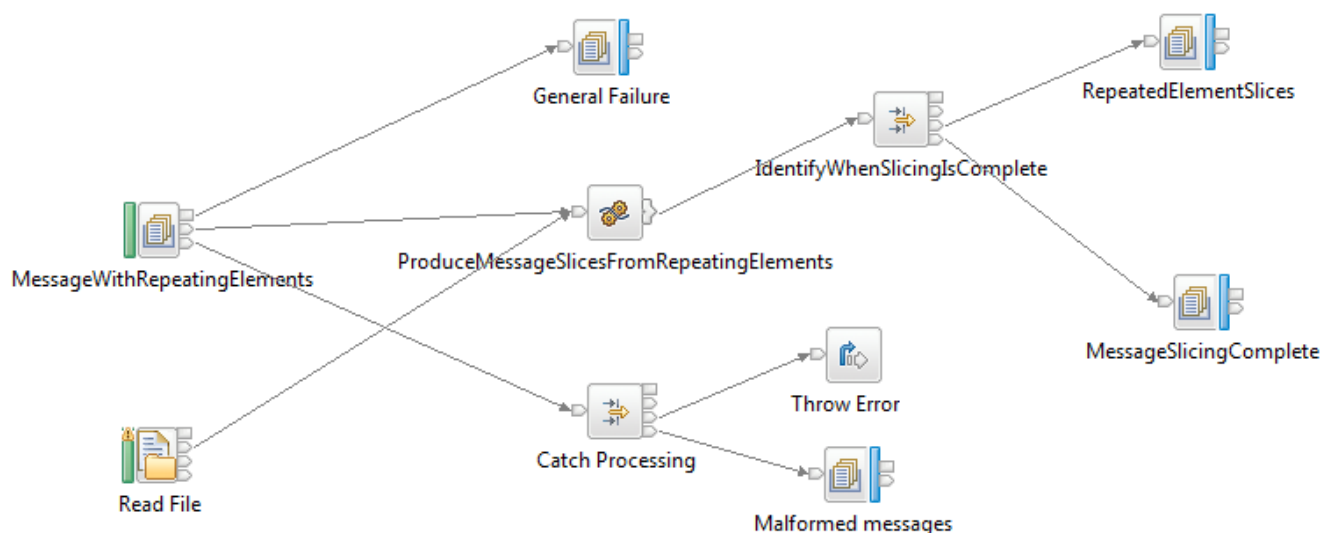
7.1 Overview

File processing is an important characteristic of an Enterprise Service Bus. (ESB). This lab will use one of the samples provided with IBM Integration Bus that shows how to read a large message and break it into multiple output messages (sometimes referred to as message shredding).

The sample takes in a large message with a repeating structure. It then processes each repeating message individually and writes an individual message for each repeating element. Since there are ten repeating elements, the input will result in ten smaller individual messages. This message flow will be modified to accept the same input in a file as well as an IBM WebSphere MQ message.

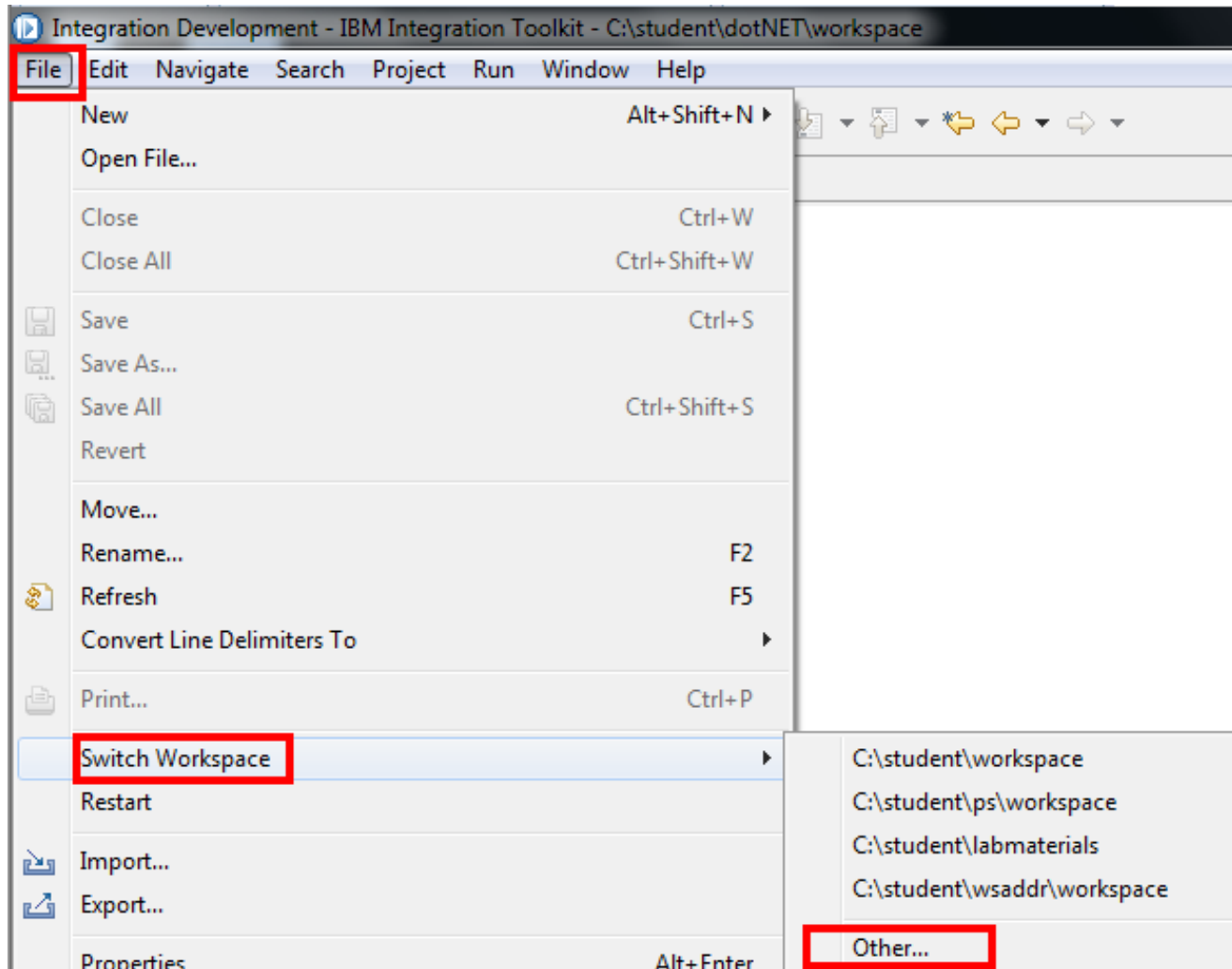
The techniques demonstrated in this sample flow show how very large files consisting of a large number of repeating segments can be processed efficiently, without requiring large amounts of memory.

The following message flow will be built.

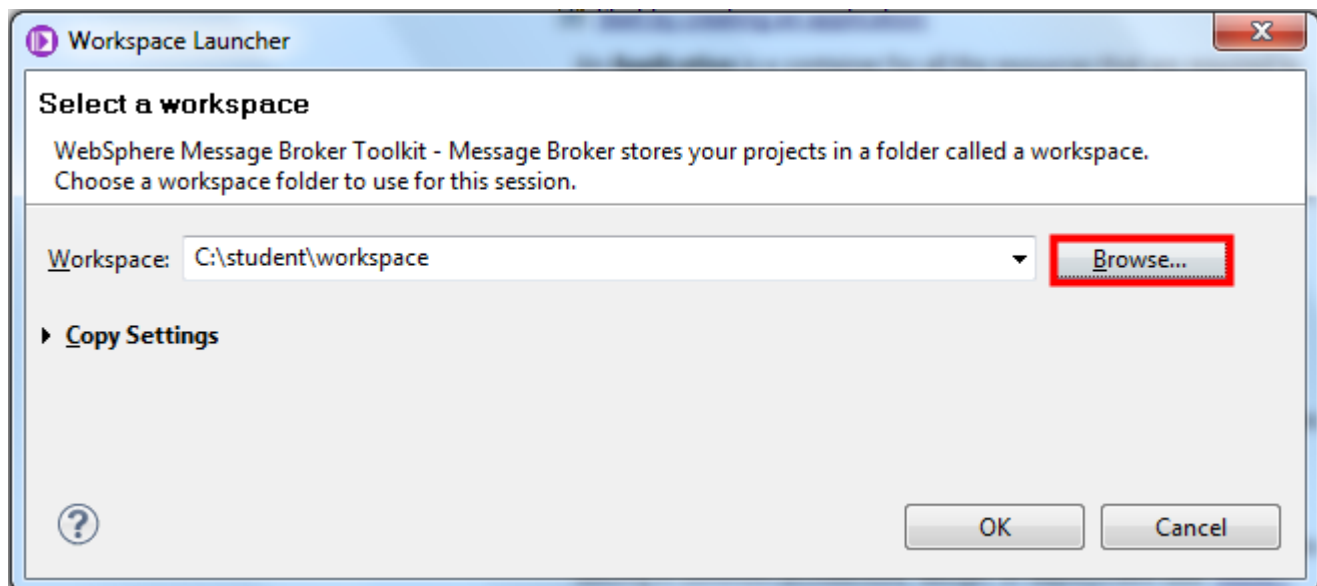


7.2 Create the message flow

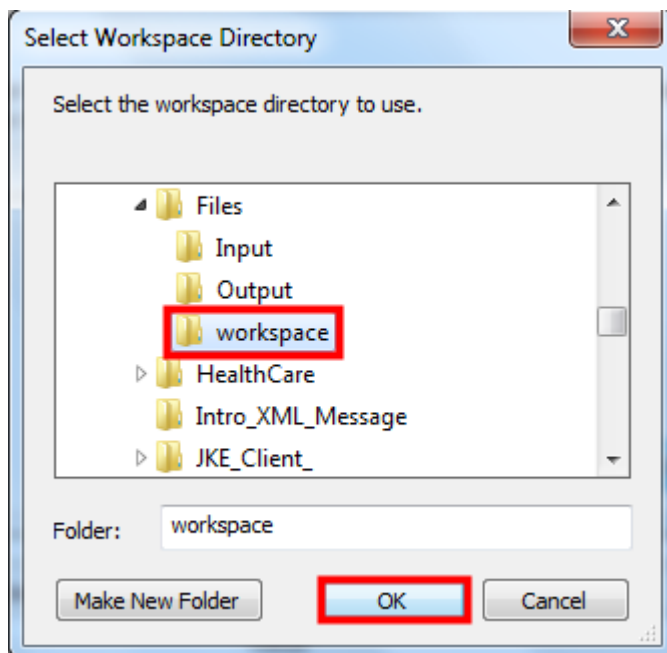
___1. Select **File**→**Switch Workspace**→**Other**.



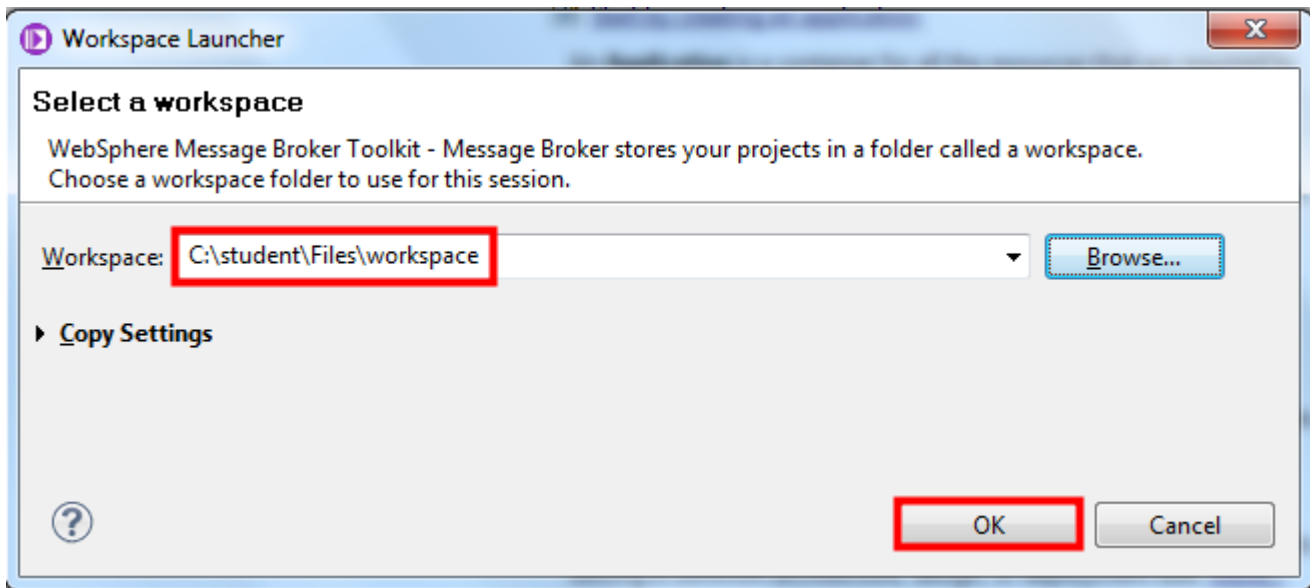
- __2. Press the **Browse** button to select the workspace.



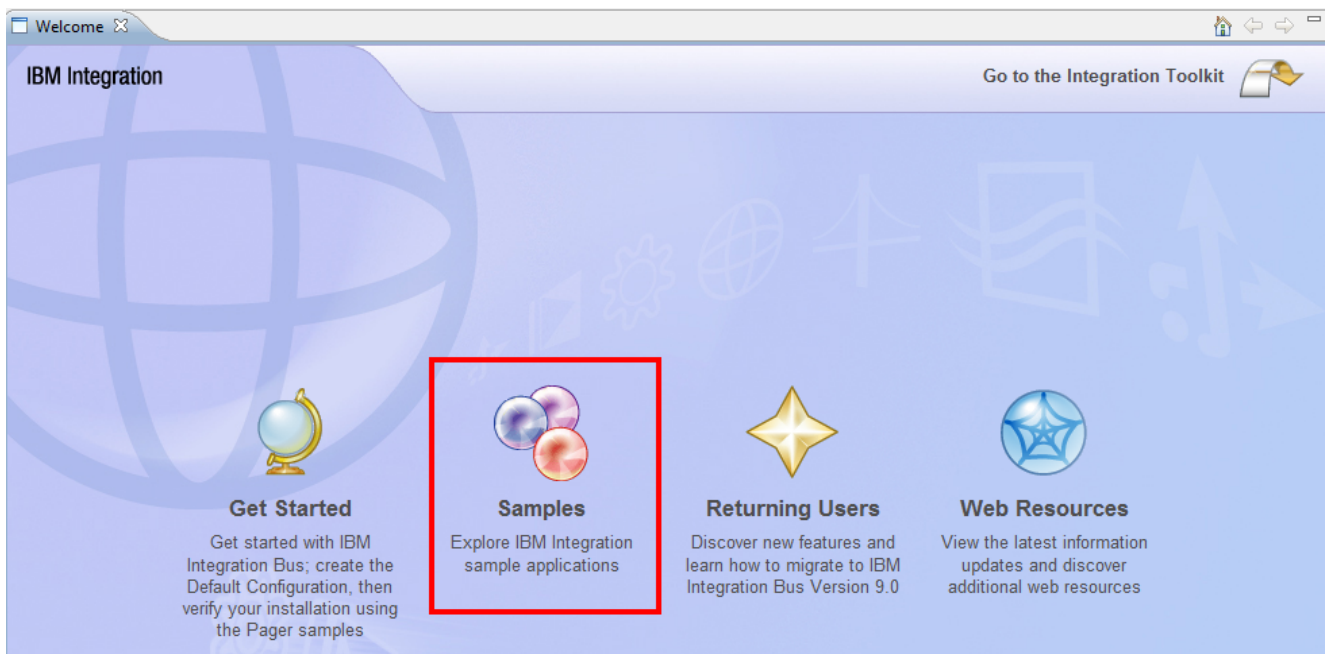
- __3. Navigate to the **C:\student\Files\workspace** directory.
- __4. Press the **OK** button to select the workspace for this lab.



__5. Press the **OK** button to open the integration toolkit.



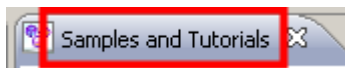
__6. Select the **Samples** icon hotspot.



__7. Close the Welcome tab.

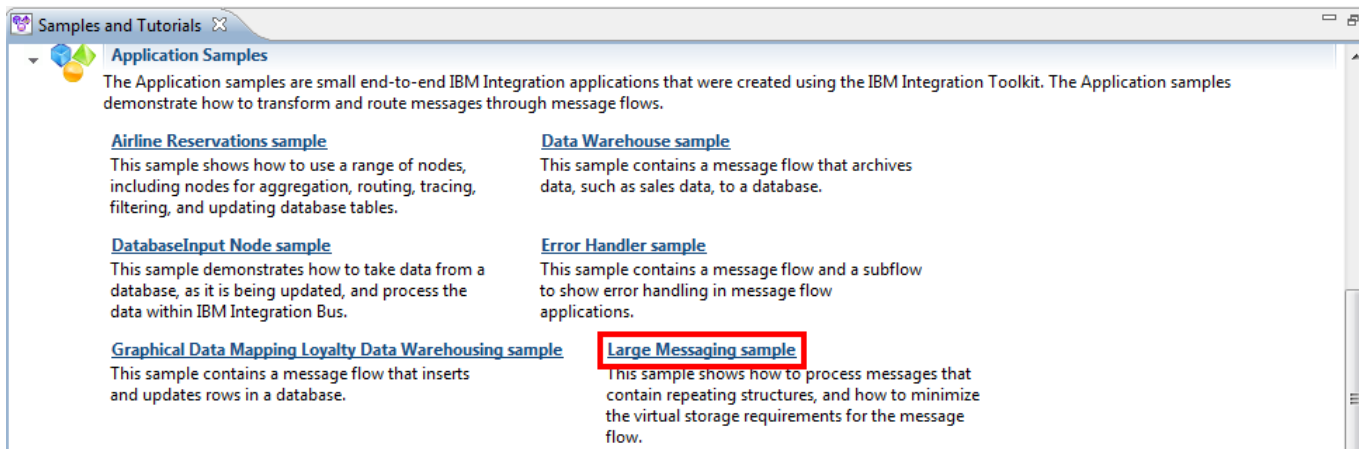


__8. Double-click the Samples and Tutorials tab to open the samples in full-screen mode.



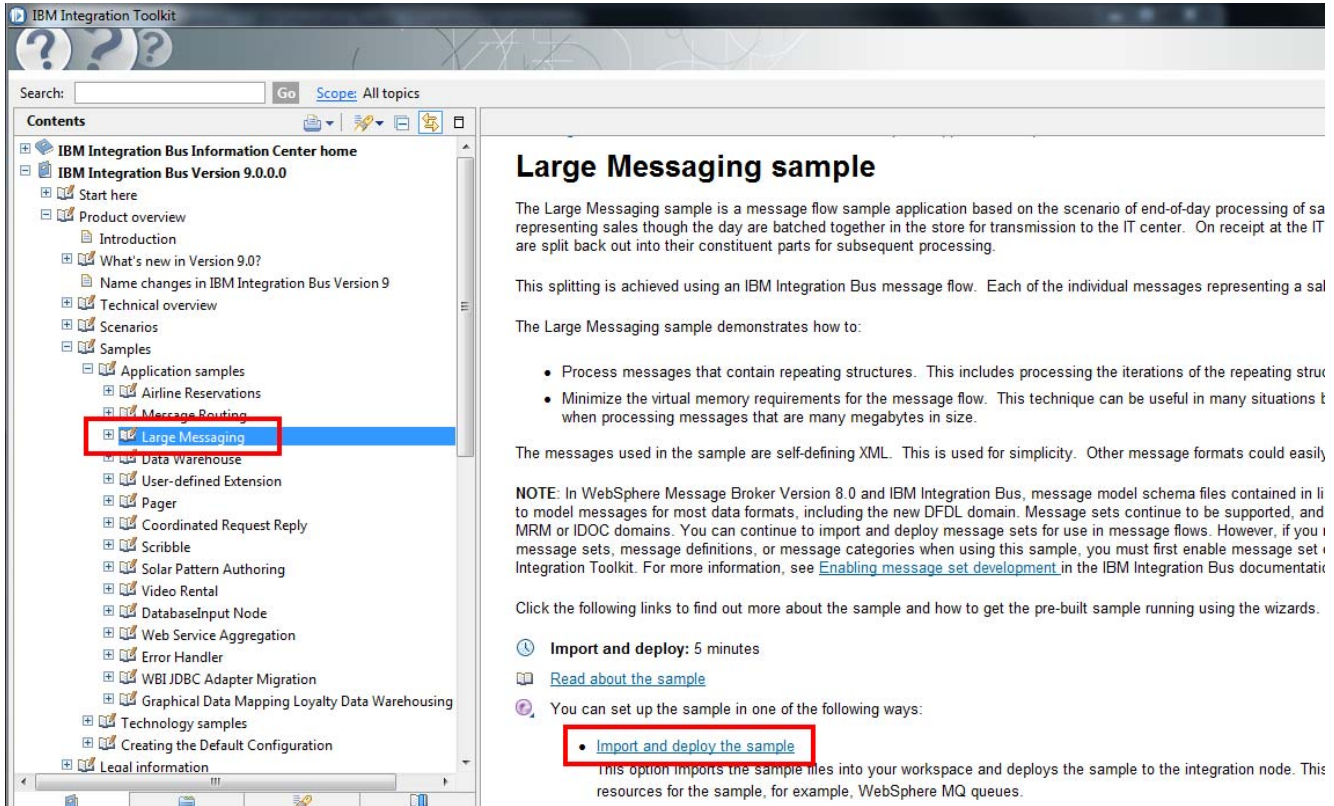
__9. Expand the **Application Samples**.

__10. Click on the **Large Messaging sample** item.



A new window should open.

__11. Select the **Import and deploy the sample** option.



The screenshot shows the IBM Integration Toolkit interface. On the left, the 'Contents' pane lists various topics under 'IBM Integration Bus Version 9.0.0.0'. The 'Large Messaging' topic is highlighted with a red box. On the right, the 'Large Messaging sample' page is displayed. It includes a description of the sample, a list of features, and instructions on how to import and deploy the sample. The 'Import and deploy the sample' link is highlighted with a red box.

Large Messaging sample

The Large Messaging sample is a message flow sample application based on the scenario of end-of-day processing of sales representing sales though the day are batched together in the store for transmission to the IT center. On receipt at the IT are split back out into their constituent parts for subsequent processing.

This splitting is achieved using an IBM Integration Bus message flow. Each of the individual messages representing a sale.

The Large Messaging sample demonstrates how to:

- Process messages that contain repeating structures. This includes processing the iterations of the repeating structure.
- Minimize the virtual memory requirements for the message flow. This technique can be useful in many situations when processing messages that are many megabytes in size.

The messages used in the sample are self-defining XML. This is used for simplicity. Other message formats could easily be used.

NOTE: In WebSphere Message Broker Version 8.0 and IBM Integration Bus, message model schema files contained in libraries to model messages for most data formats, including the new DFDL domain. Message sets continue to be supported, and MRM or IDOC domains. You can continue to import and deploy message sets for use in message flows. However, if you use message sets, message definitions, or message categories when using this sample, you must first enable message set development in the Integration Toolkit. For more information, see [Enabling message set development](#) in the IBM Integration Bus documentation.

Click the following links to find out more about the sample and how to get the pre-built sample running using the wizards.

- [Import and deploy: 5 minutes](#)
- [Read about the sample](#)

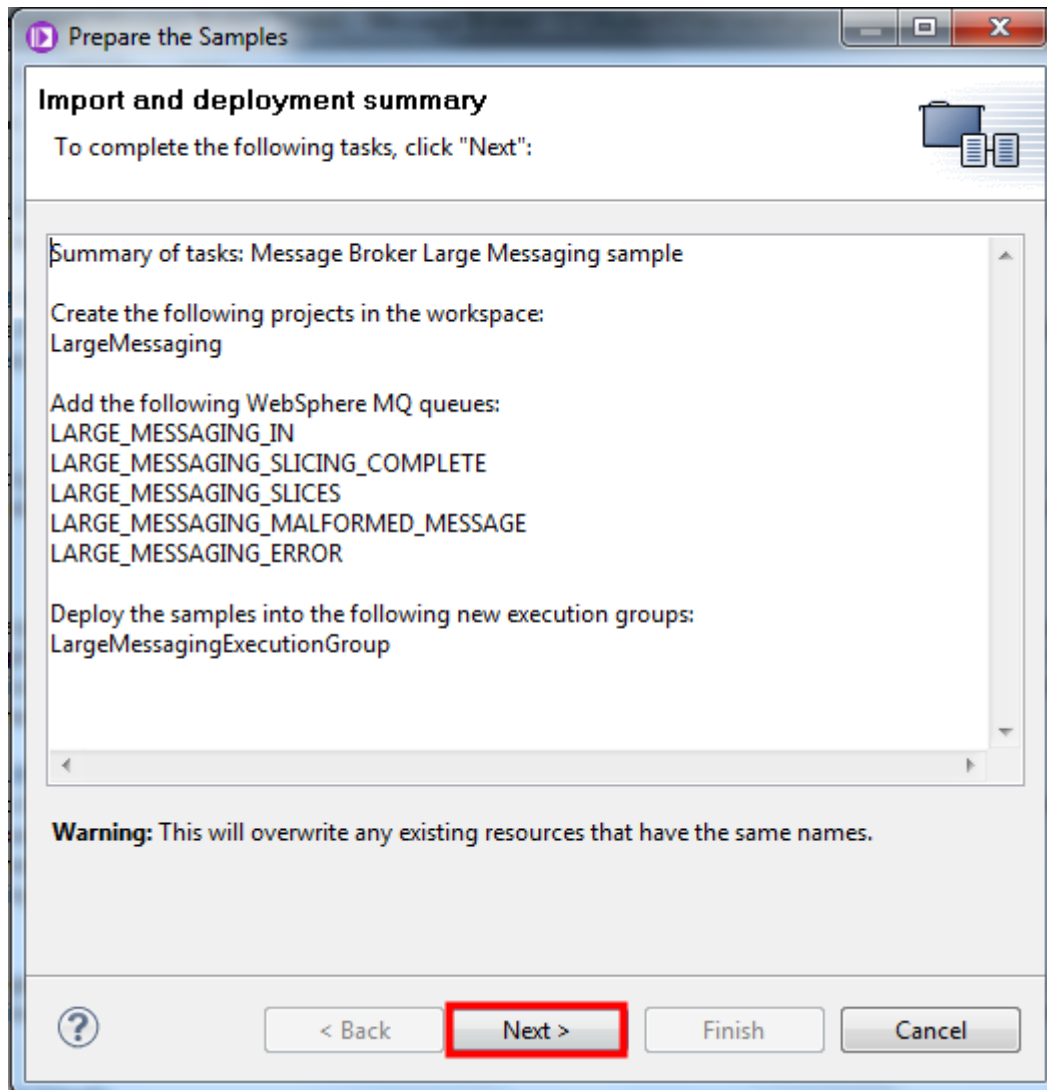
You can set up the sample in one of the following ways:

- [Import and deploy the sample](#)

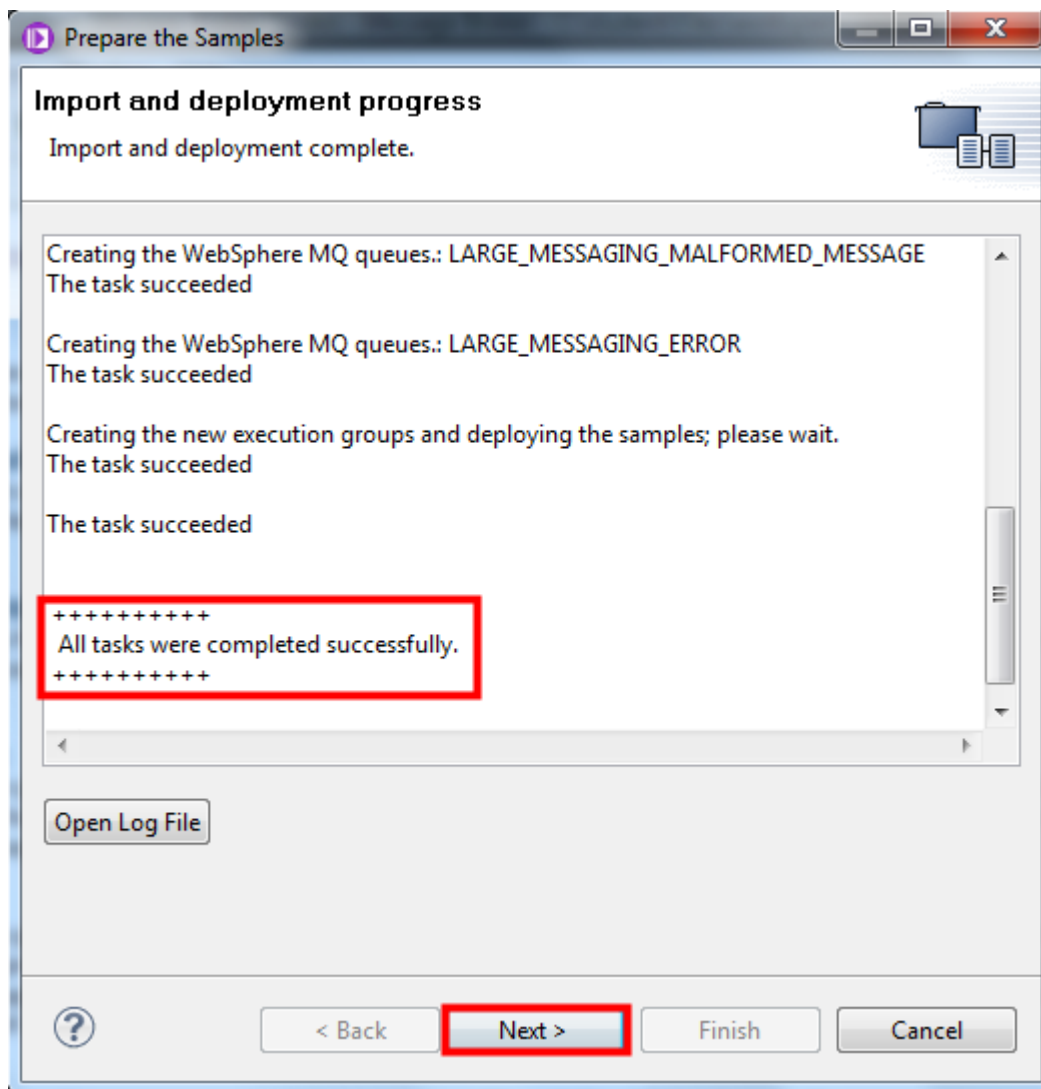
This option imports the sample files into your workspace and deploys the sample to the integration node. This option also creates the resources for the sample, for example, WebSphere MQ queues.

An **Import and deployment summary** dialog should appear.

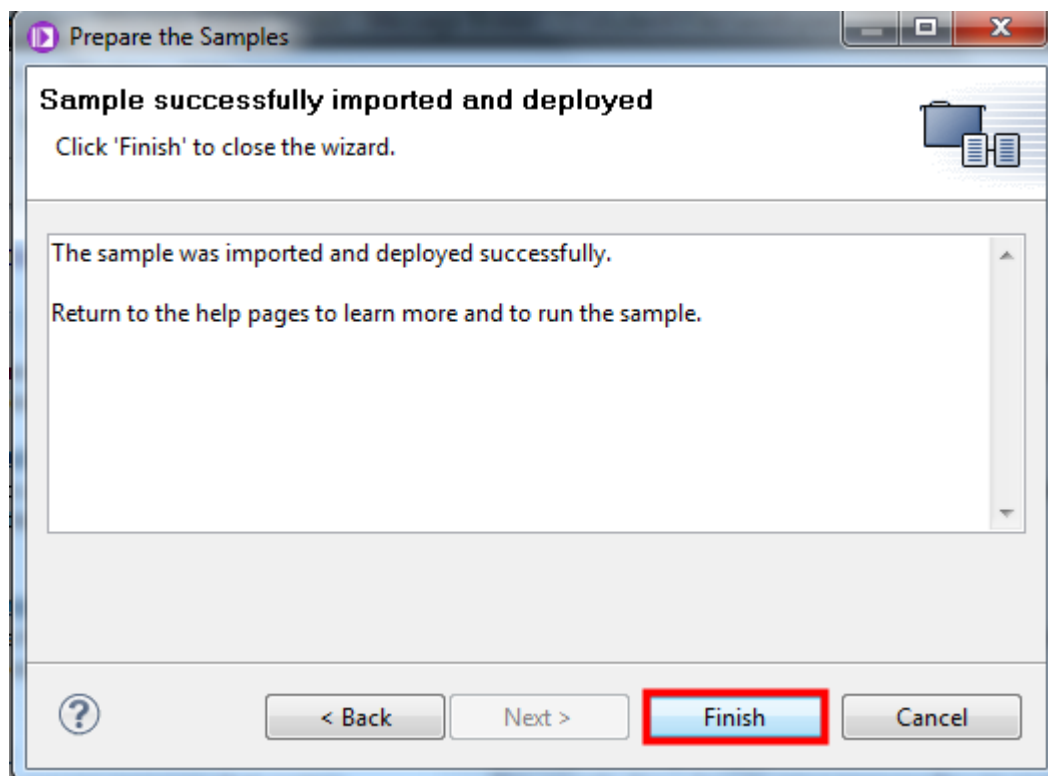
- __12. Press the **Next** button to import the sample project, create the queues and deploy the message flow.



- __13. Scroll down and confirm that all the tasks were successful.
- __14. Press the **Next** button to continue.



__15. Press the **Finish** button to dismiss the dialog.

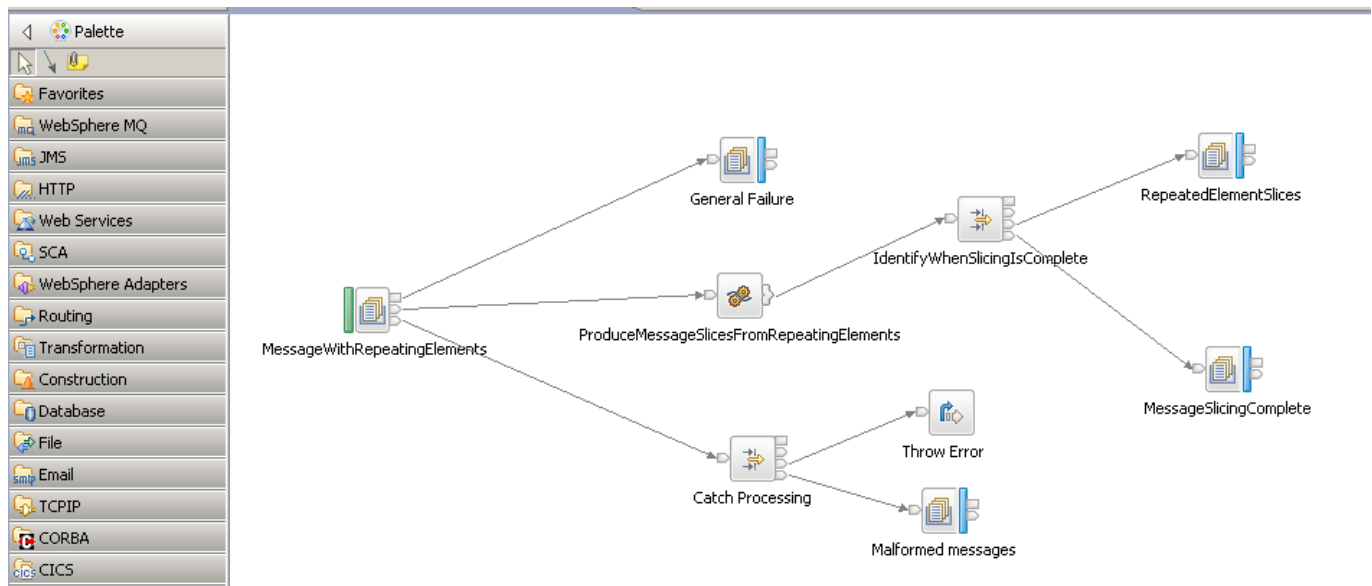


Take some time to read about the sample.

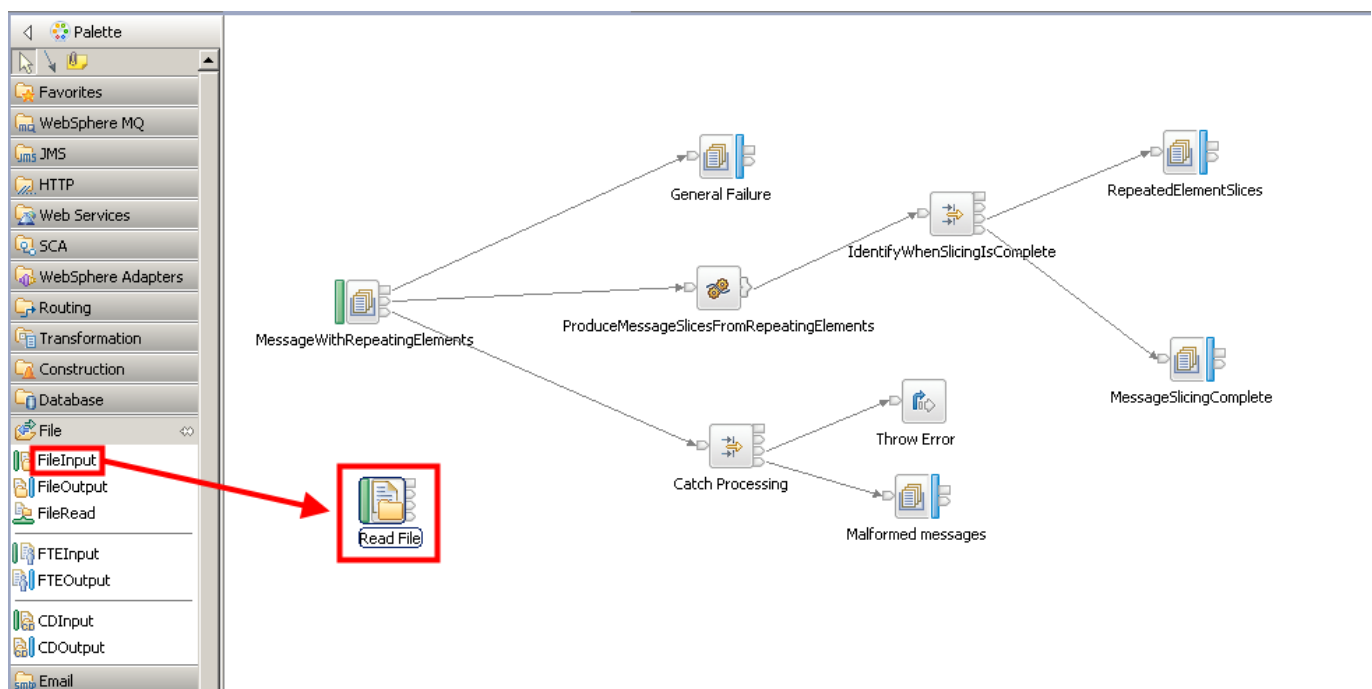
__16. Close the **Information Center** window.



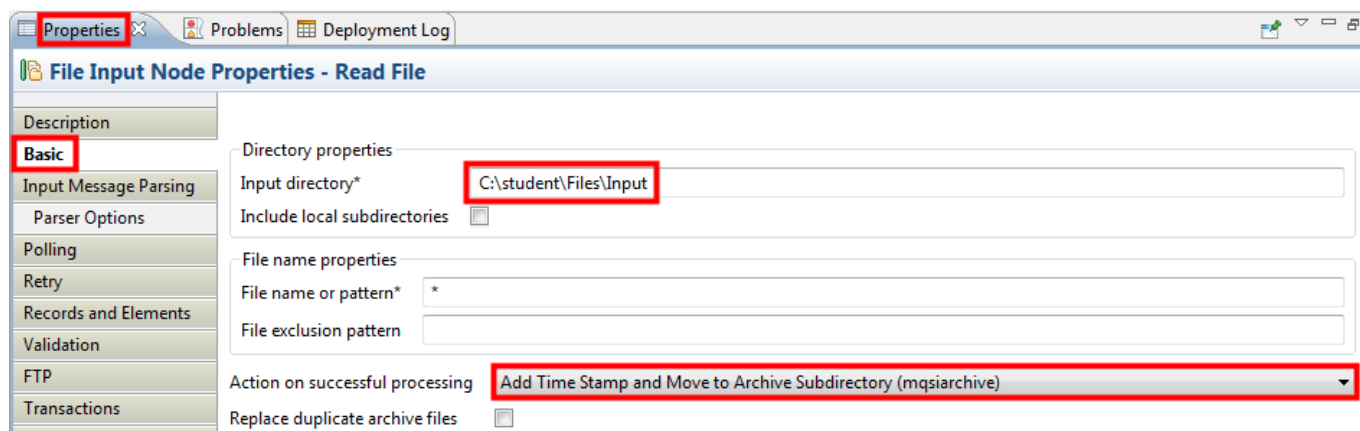
The imported message flow should be open in the message flow editor.



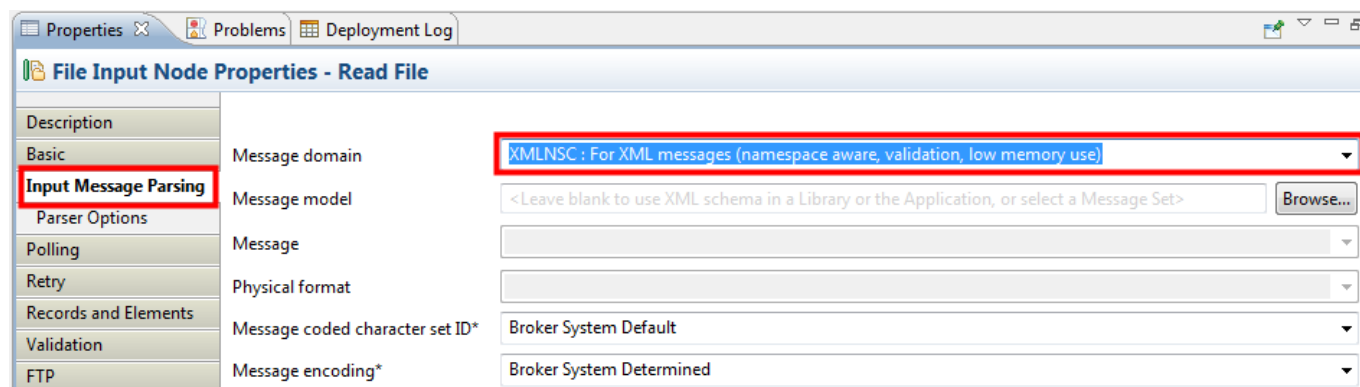
- __17. Expand the **File** folder.
- __18. Select the **FileInput** node.
- __19. Drag it to the canvas and press the left mouse button to add the node to the message flow.
- __20. Change the name of the new node to **Read File**.
- __21. Press the **Enter** key to complete the rename operation.



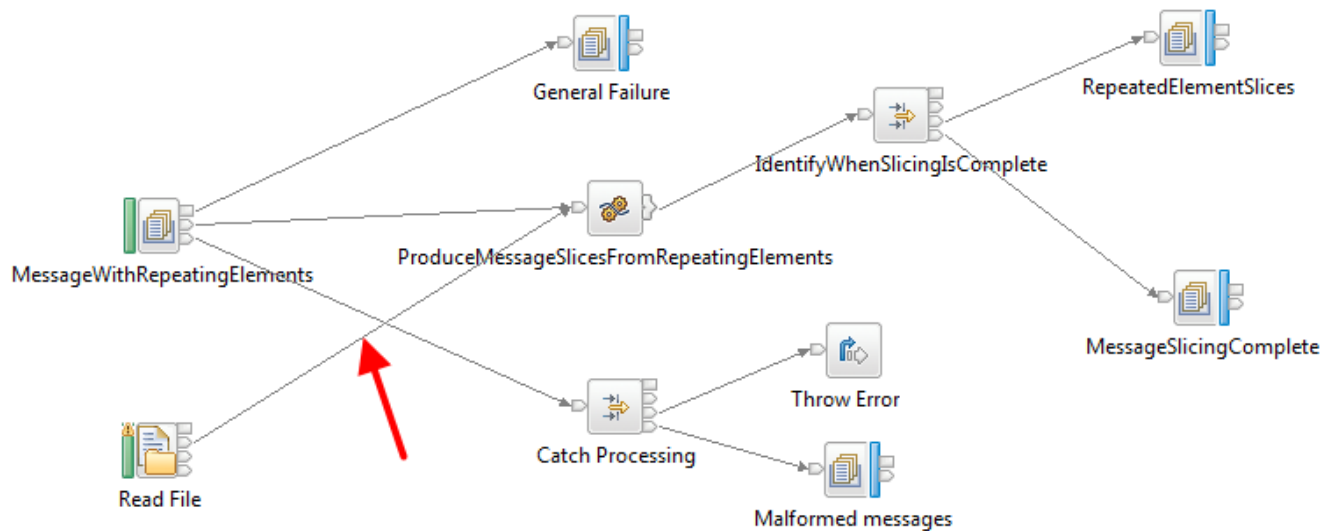
- __22. If the flow editor is in full-screen mode double-click the **Read File** node to display the properties.
- __23. In the **Properties** pane select the **Basic** tab.
- __24. Enter **C:\student\Files\Input** as the **Input directory**.
- __25. Use the drop-down menu to select the **Add Time Stamp and Move to Archive Subdirectory (mqsiarchive)** as the **Action on successful processing**.



- __26. Select the **Input Message Parsing** tab.
- __27. Use the drop-down menu to select the **XMLNSC** parser for the **Message domain**.

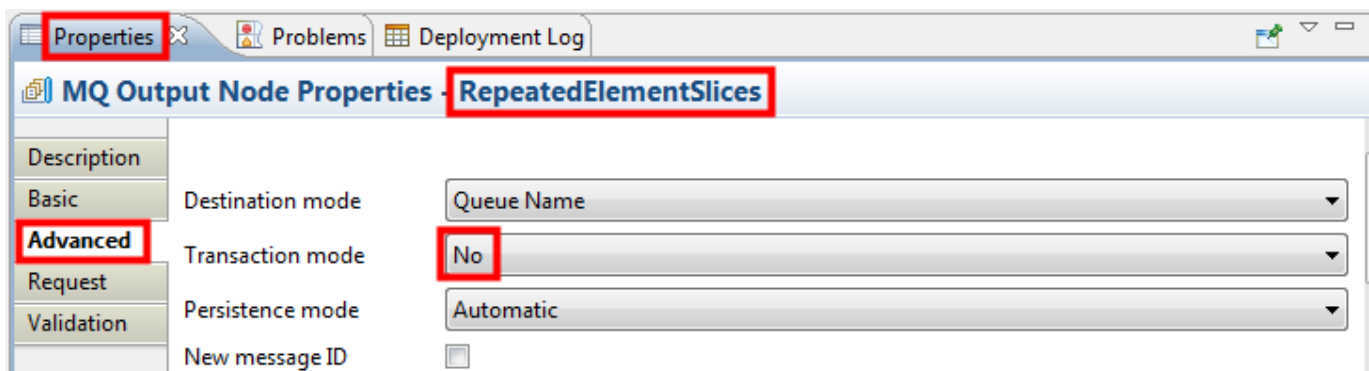


- __28. Wire the **Out** terminal of the **Read File** node to the **In** terminal of the **ProduceMessageSlicesFromRepeatingElements** node.



One more change will be made to allow for a very large file. The processing of the sample is based on MQ messages and as such the output messages are part of a single unit of work. However, this would not scale to a very large file. The “repeated element slices” messages will therefore be written outside of a sync point. In a production environment this could lead to duplicate messages in the event of a failure. The duplicate messages could be easily handled by using a resequence node and putting a sequence number in the local environment. This would eliminate duplicate messages resulting from a failure.

- __29. Select the **RepeatedElementSlices** node.
- __30. In the **Properties** for the node select the **Advanced** tab.
- __31. Use the drop down menu to select **No** as the **Transaction mode**.



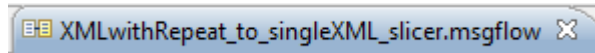
- __32.  Save the message flow.

The message flow is now complete. It has been modified to accept files as well as messages as input.

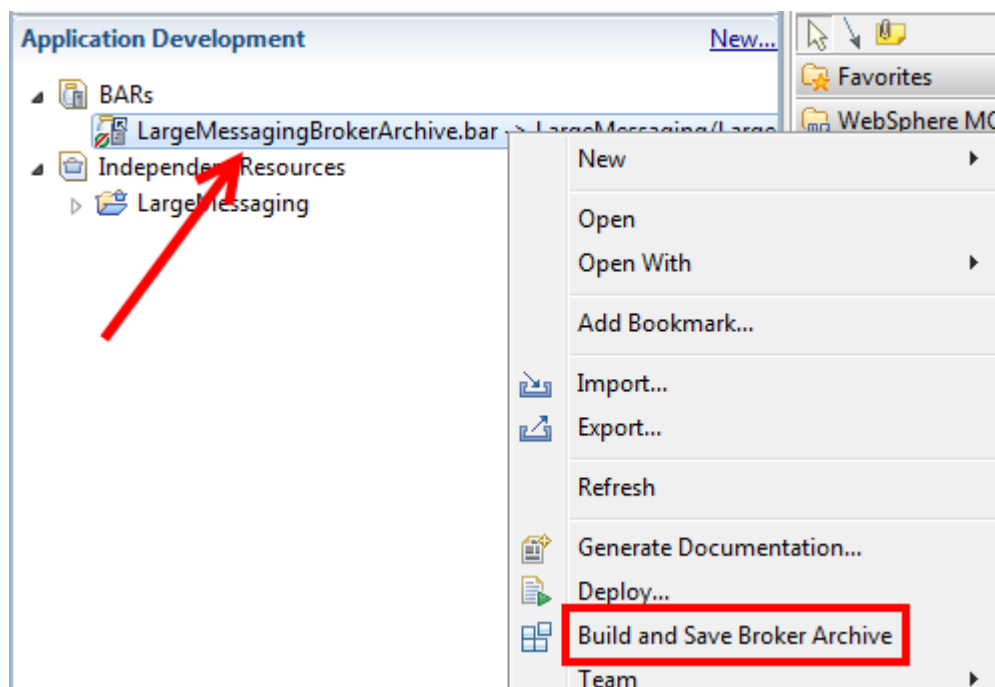
7.3 Execute the message flow

The message flow will be tested using both a message and a file as inputs. The message path will be tested first. The sample program includes a flow test with data. This will be used to test the messaging path.

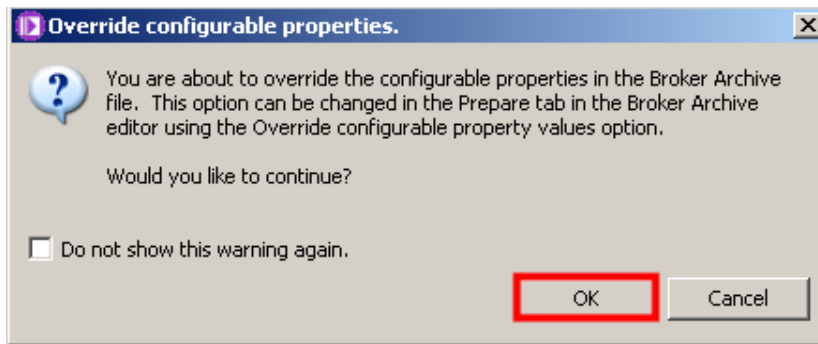
- ___1. If the screen is in full-screen mode double-click the message flow editor tab to return to windowed mode.



- ___2. In the navigator pane, expand **Independent Resources**→**LargeMessaging**→**BARs**.
- ___3. Select the **LargeMessagingBrokerArchive.bar** broker archive file.
- ___4. Press the right mouse button.
- ___5. Select **Build and Save Broker Archive** from the menu.



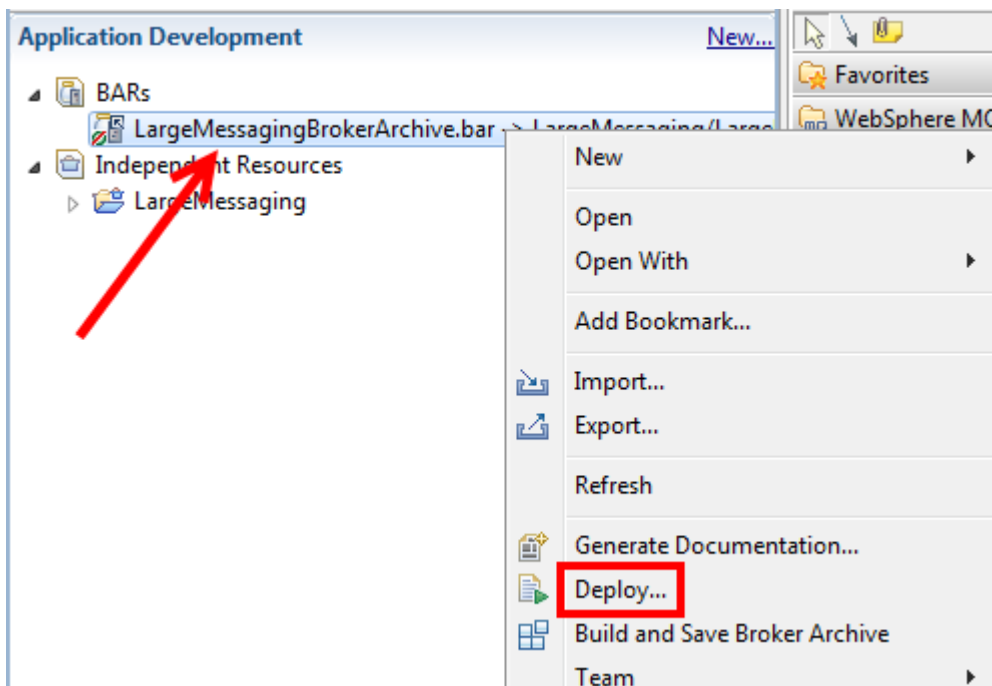
- __6. Press the OK button to dismiss the warning (if it appears).



- __7. In the Project Navigator pane select the **LargeMessagingBrokerArchive.bar** archive file.

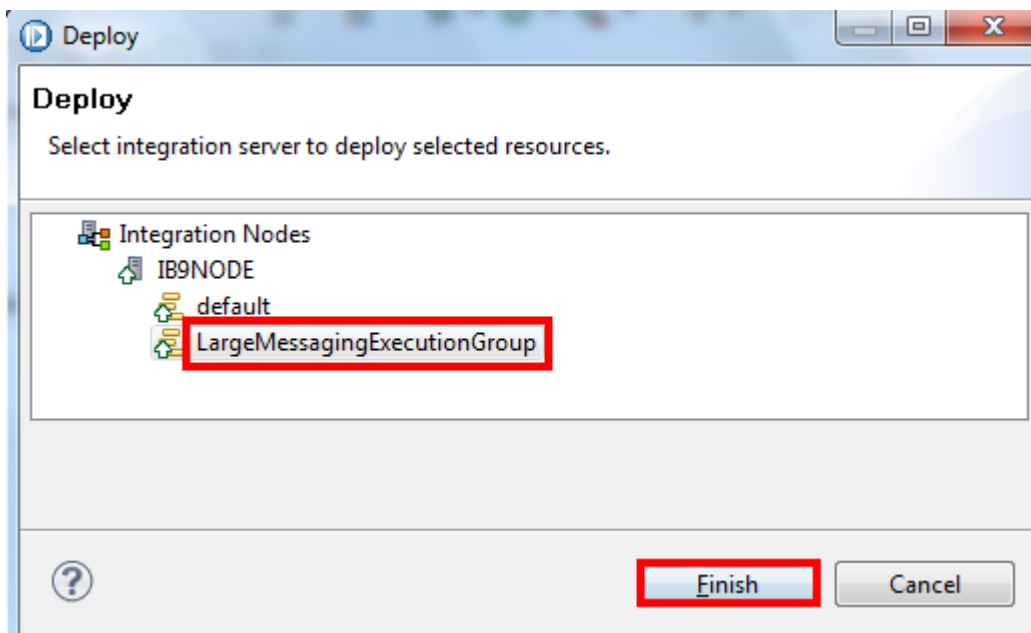
- __8. Press the right mouse button.

- __9. Select **Deploy...** from the menu.

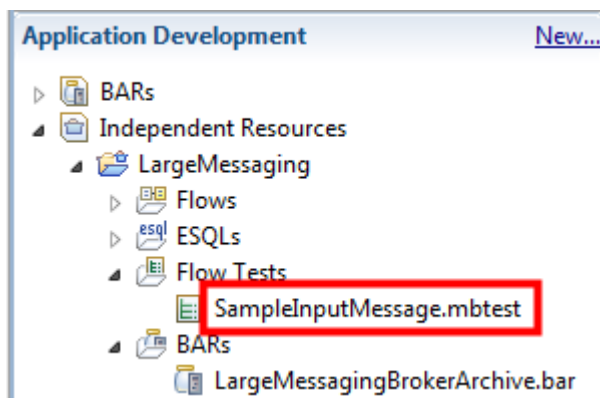


The import and deploy wizard created an additional integration server where it deployed the original version of the message flow. The updated message flow will be deployed to replace the original version.

- __10. Expand **Integration Nodes**→**IB9NODE**.
- __11. Select the **LargeMessagingExecutionGroup** integration server.
- __12. Press the **Finish** button to initiate the deployment operation.

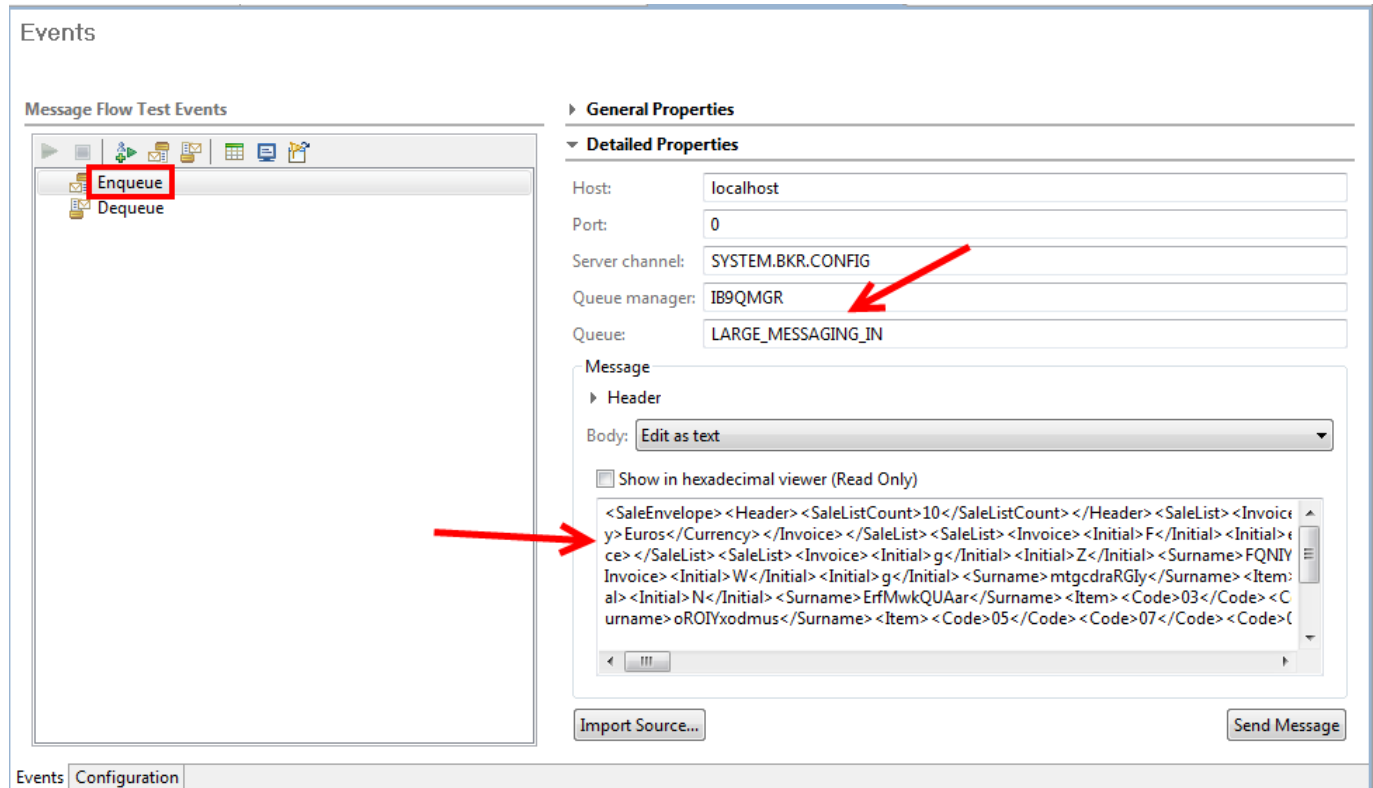


- __13. In the navigator pane, expand **LargeMessaging**→**Flow Tests**.
- __14. Double-click on the **SampleInputMessage.mbttest** file to launch the Test Client.



Examine the test client window. The **Enqueue** entry should be selected. This will write a message to the specified queue. The name of the queue and queue manager is specified under **Detailed Properties**. The input data is shown in the large text box in the **Message** area.

__15. Press the **Send Message** button to send the data to the specified queue.



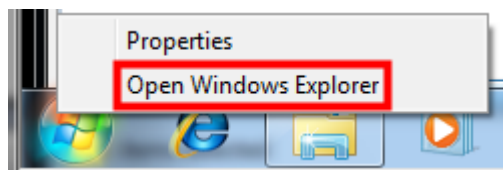
The results of the execution of the message flow will now be examined.

- __16. Return to MQ Explorer. If it is not running start it.
- __17. Expand the **IB9QMGR** queue manager.
- __18. Select the **Queues** folder.
- __19. In the right-hand pane scroll down so the **LARGE_MESSAGING_SLICES** queue is visible.
- __20. Confirm that there are ten messages in the **LARGE_MESSAGING_SLICES** queue.
- __21. Confirm that there is one message in the **LARGE_MESSAGING_SLICING_COMPLETE** queue.

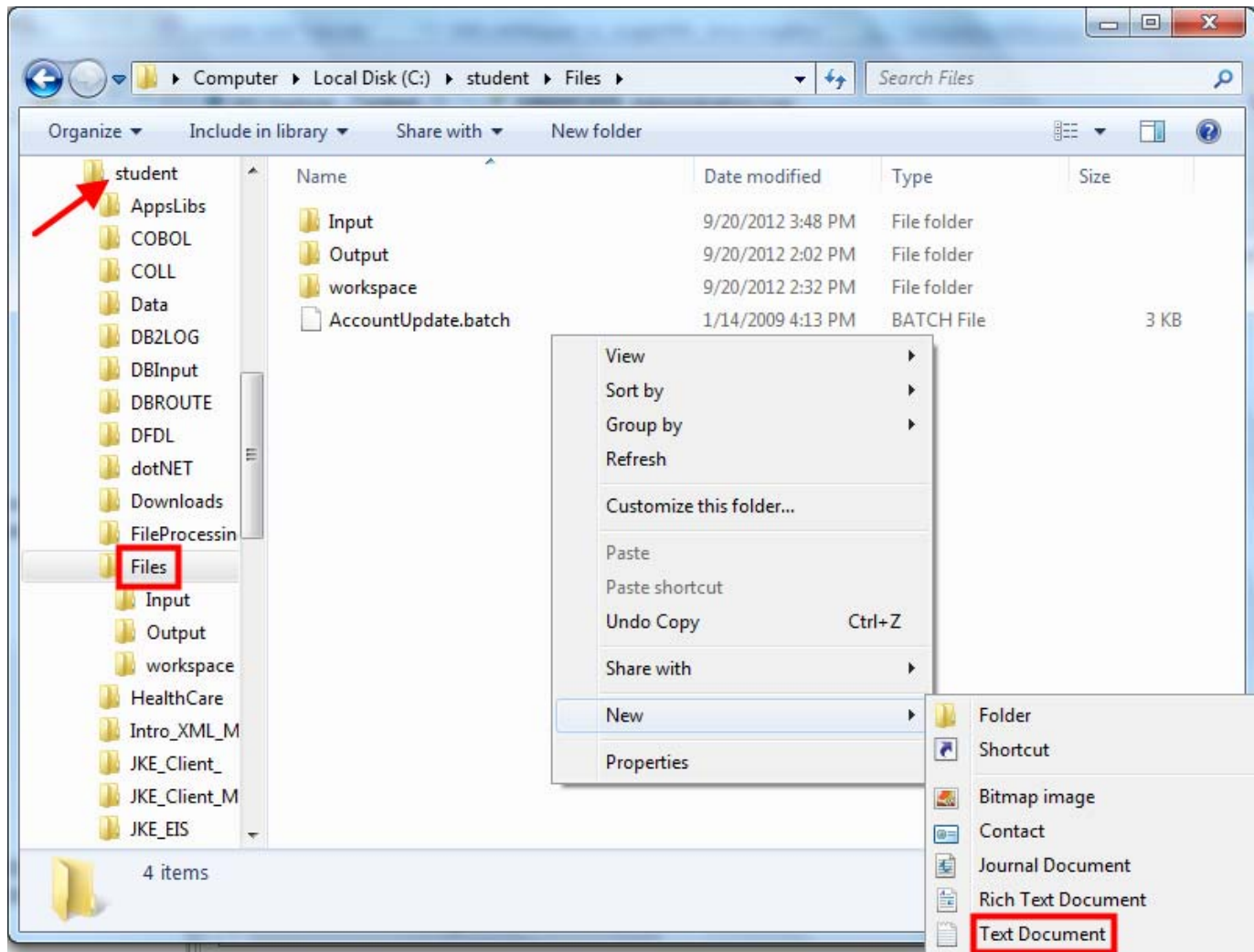
Queues				
Filter: Standard for Queues				
Queue name	Queue type	Open input count	Open output count	Current q
LAB.STATE	Local	0	0	0
LAB.US.OUT	Local	0	0	0
LARGE_MESSAGING_ERROR	Local	0	0	0
LARGE_MESSAGING_IN	Local	1	0	0
LARGE_MESSAGING_MALFORMED_MESSAGE	Local	0	0	0
LARGE_MESSAGING_SLICES	Local	0	1	10
LARGE_MESSAGING_SLICING_COMPLETE	Local	0	1	1
MAP.ACCTCLOSE	Local	0	0	0

The same test data will now be copied into a file and the file path will be tested. A Windows Explorer session will be used.

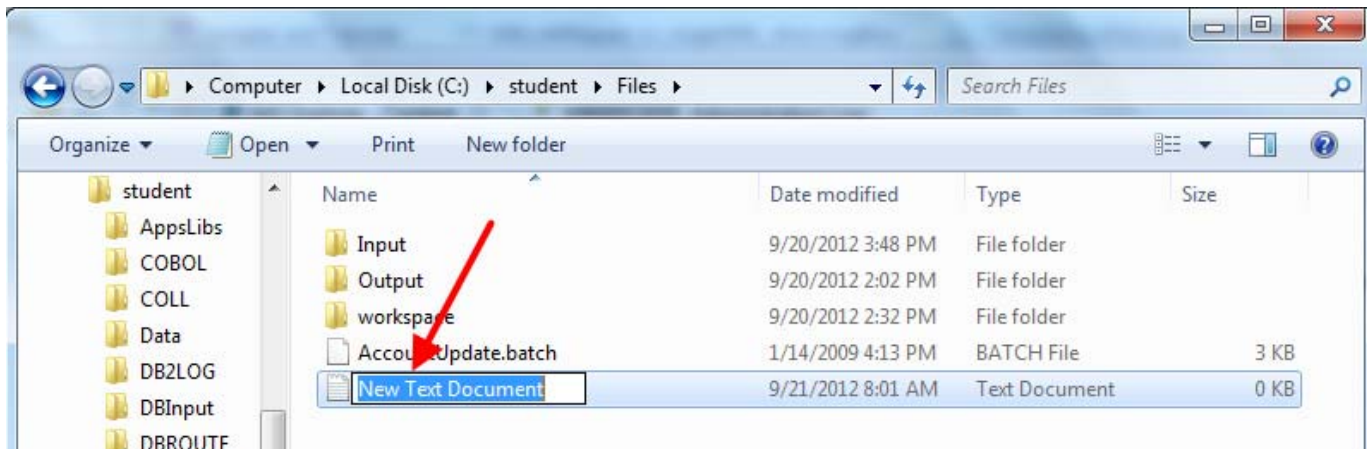
- __22. Select the **Windows Start button**.
- __23. Press the right mouse button.
- __24. Select **Open Windows Explorer**.



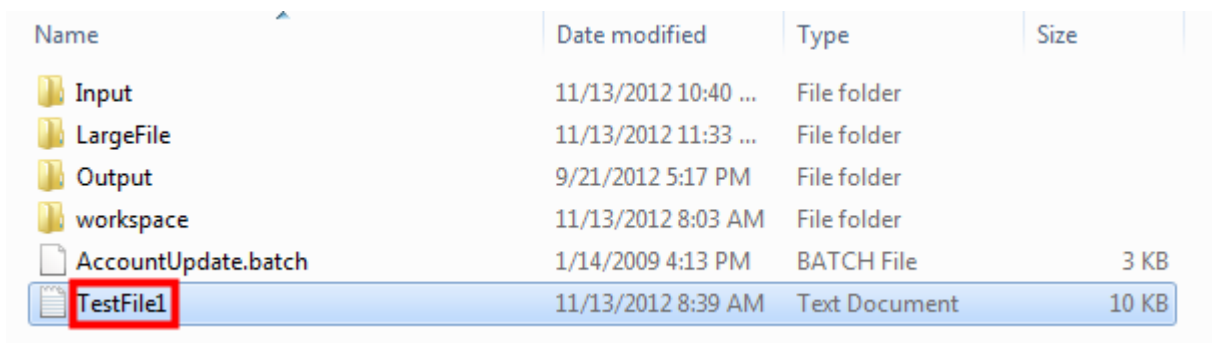
- __25. Navigate to the **C:\student\Files** directory.
- __26. Select the **Files** directory.
- __27. Point to a blank area in the right hand pane and press the right mouse button.
- __28. Select **New→Text Document** from the menu that appears.



A new file should be created in the **Files** folder.



- __29. Change the name of the new file to **TestFile1**.
- __30. Press the **Enter** key to complete the rename operation.
- __31. Double-click the new **TestFile1** file to open the file in the notepad editor.



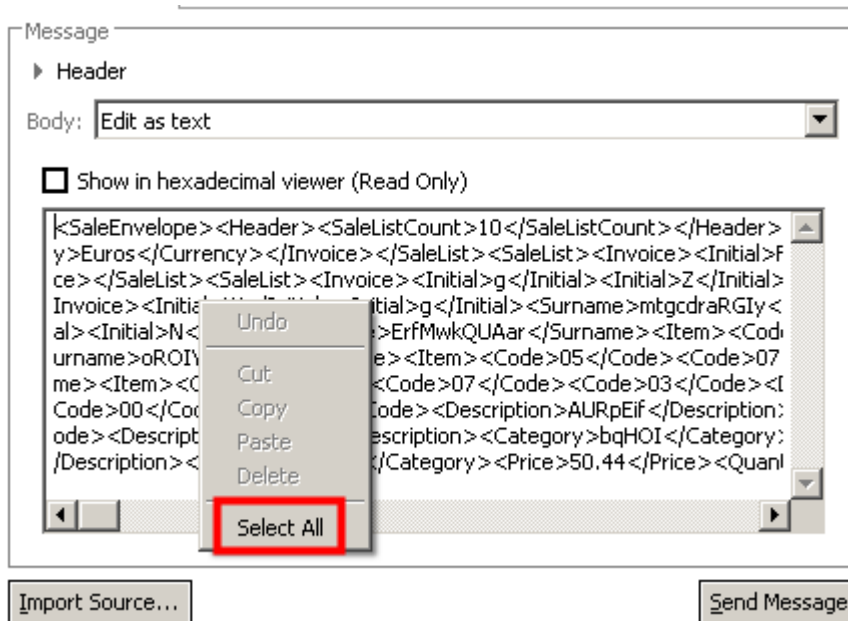
A notepad session should open. The file is currently empty. The test data from the test client will now be copied and pasted into the notepad session.

__32. Return to the test client in the Integration Toolkit.

__33. Move the mouse pointer to the message text.

__34. Press the right mouse button.

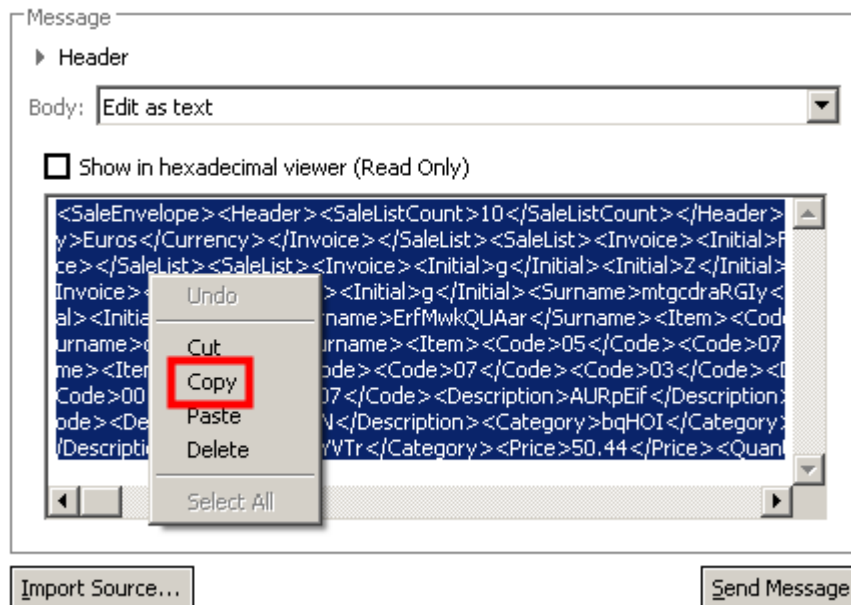
__35. Click the **Select All** menu item.



__36. Move the mouse pointer to the message text.

__37. Press the right mouse button.

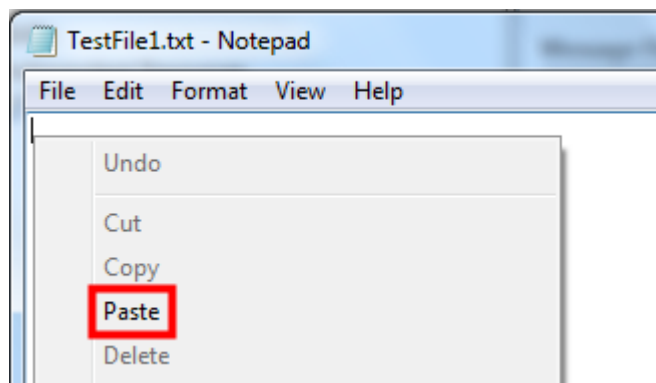
__38. Click the **Copy** menu item.



__39. Return to the notepad session.

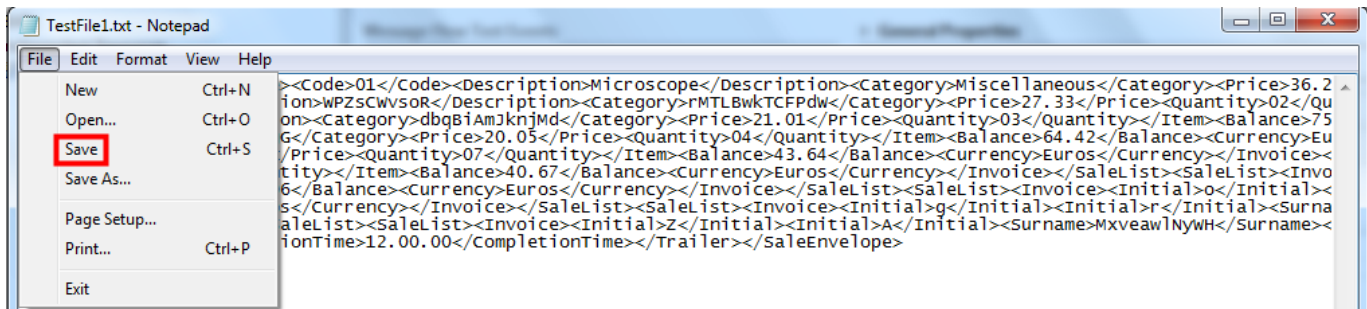
__40. Point the mouse in a blank area of the notepad session and press the right mouse button.

__41. Select **Paste** from the menu.

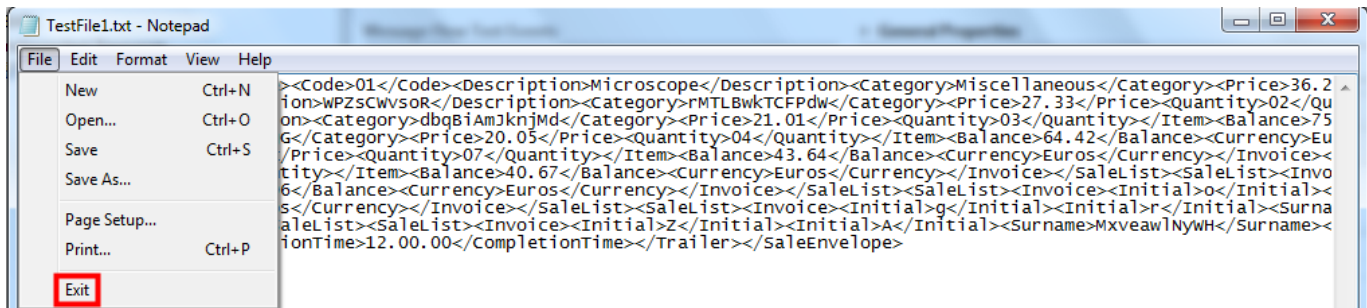


The data should now have been transferred into the notepad editor.

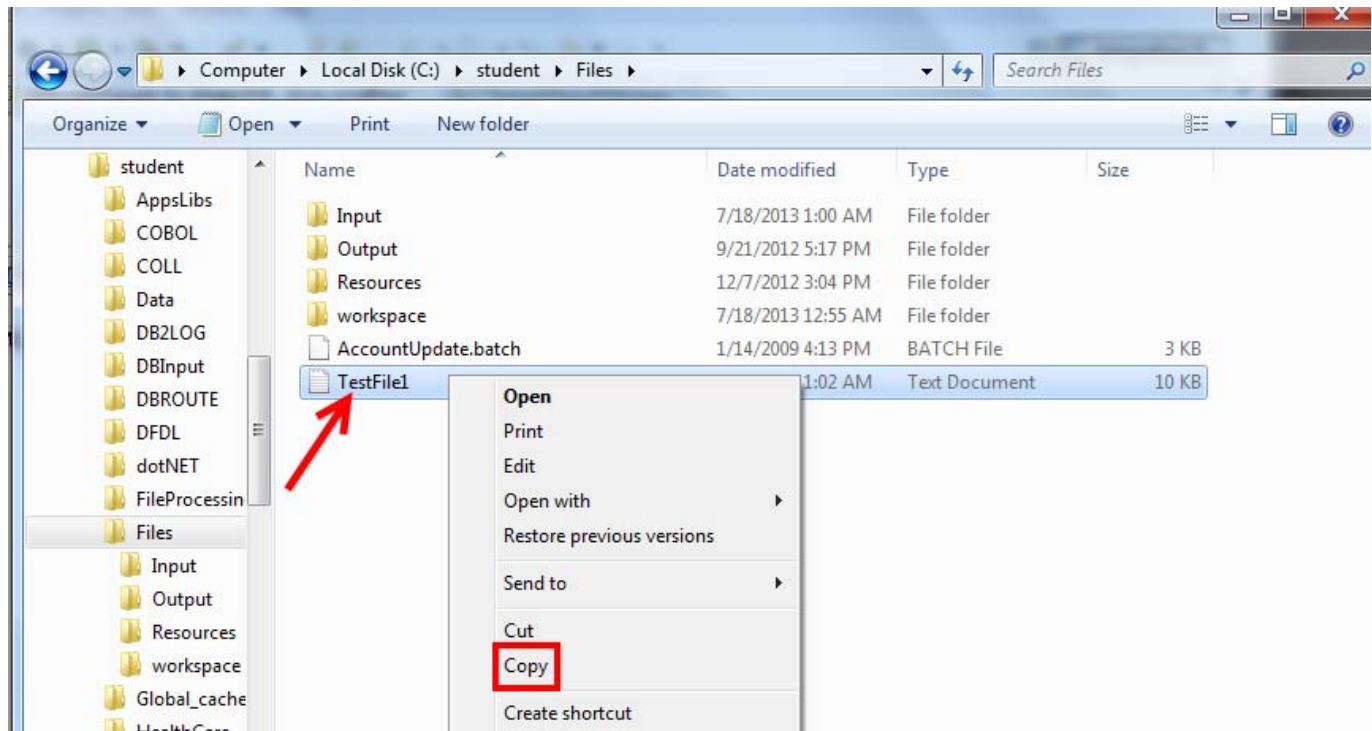
__42. Select **File→Save** from the menu in the notepad editor (or use **Ctrl+S**).



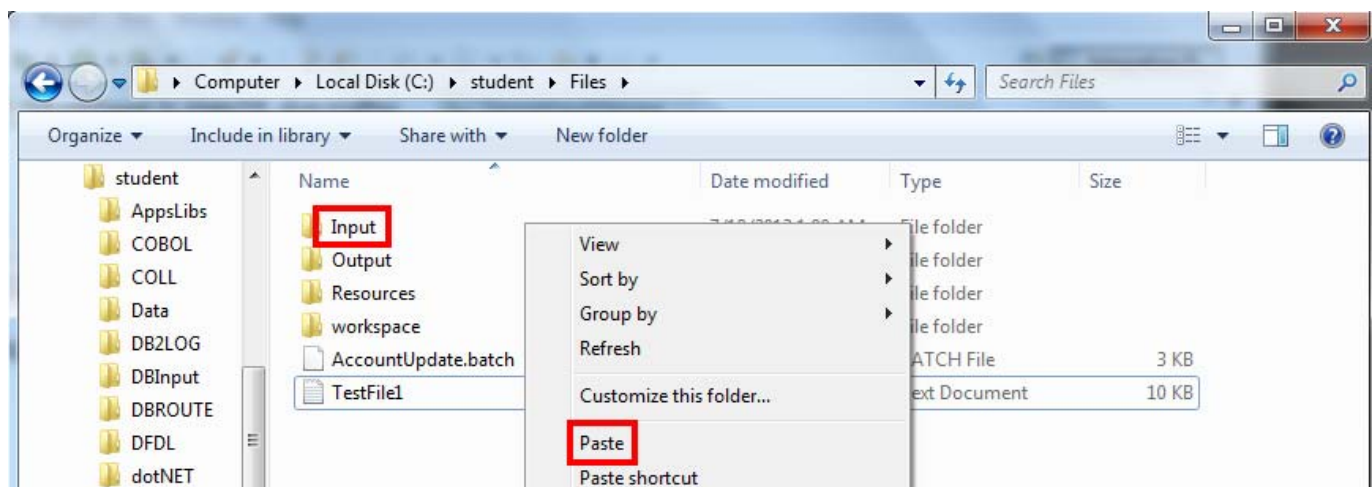
__43. Close the notepad editor (either **Alt+F4** or **File→Exit**).



- __44. Return to the Windows Explorer window. The new **TestFile1.txt** file should be visible.
- __45. Select the **TestFile1.txt** file.
- __46. Press the right mouse button.
- __47. Select **Copy** from the menu.



- __48. Select the **Input** directory above the file.
- __49. Press the right mouse button.
- __50. Select **Paste** from the menu



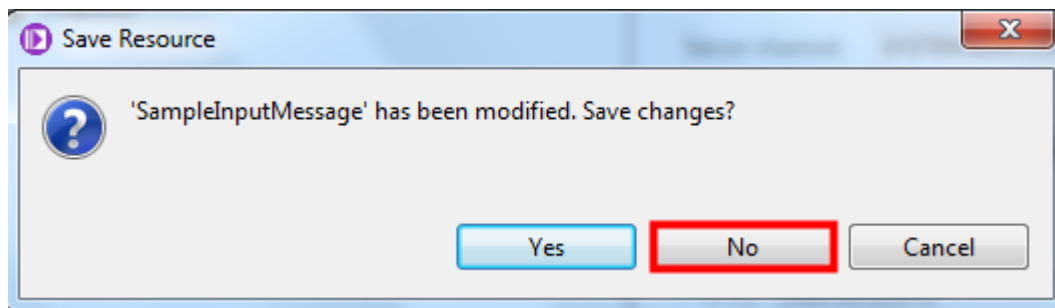
- __51. Return to the MQ Explorer and wait for a refresh cycle.
- __52. The queue depth of the **LARGE_MESSAGING_SLICES** queue should now be twenty.

Queues				
Filter: Standard for Queues				
Queue name	Queue type	Open input count	Open output count	Current q
LAB.STATE	Local	0	0	0
LAB.US.OUT	Local	0	0	0
LARGE_MESSAGING_ERROR	Local	0	0	0
LARGE_MESSAGING_IN	Local	1	0	0
LARGE_MESSAGING MALFORMED MESSAGE	Local	0	0	0
LARGE_MESSAGING_SLICES	Local	0	2	20
LARGE_MESSAGING_SLICING_COMPLETE	Local	0	2	2
MAP.ACCTCLOSE	Local	0	0	0

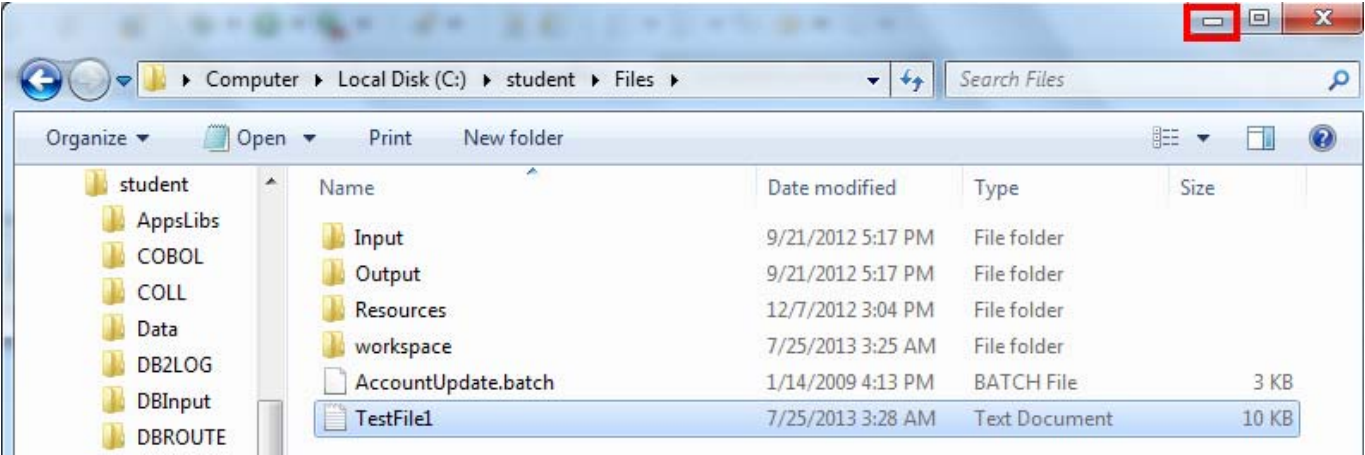
- __53. Return to the Integration Toolkit.
- __54. **Close** the **Test Client**.



- __55. Press the **No** button. The changes to the test client will not be saved.



__56. Minimize the Windows Explorer window.



7.4 Test with a very large file

This section is optional. A much larger version of the test file will be created. In order to reduce the size of the VMWare image the file itself will be created using a small program. Please do not try to examine it using a simple editor such as Notepad or WordPad, as it is too large. The program should take about a minute to create the large file.

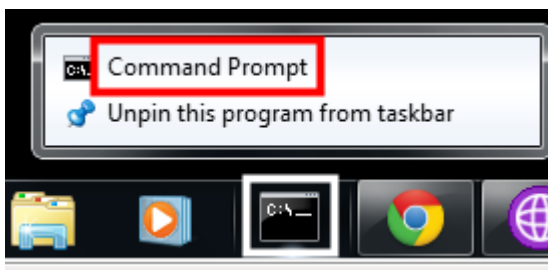
A program has been provided to create a large file with a specified number of records. Two DOS command files are provided. One will create a file approximately 1 GB in size. The other will create a file approximately 100 MB in size. The larger file (1 GB) will run approximately ten times longer. On a typical laptop the 100 MB file will take approximately one minute to process, while the 1 GB file will take approximately 10 minutes to process. The message flow will be executed using this large file. The file contains the same type of data as the test file that was used but with many more repeating elements.

The other consideration is the size of the output queue. The maximum depth of the output queue is 5000 messages. The input file contains many more records than the queue can hold. In a production environment the maximum number of messages would have to be increased. However, in this case, a program will be run that will both drain the output messages as they are produced and report the rate that messages are being read. This will prevent overflowing the queue. On the author's laptop, messages are created at about 1500-1600 per second. The 100MB file will contain 100,500 individual records and the 1 GB file will contain 1,005,000 individual records.

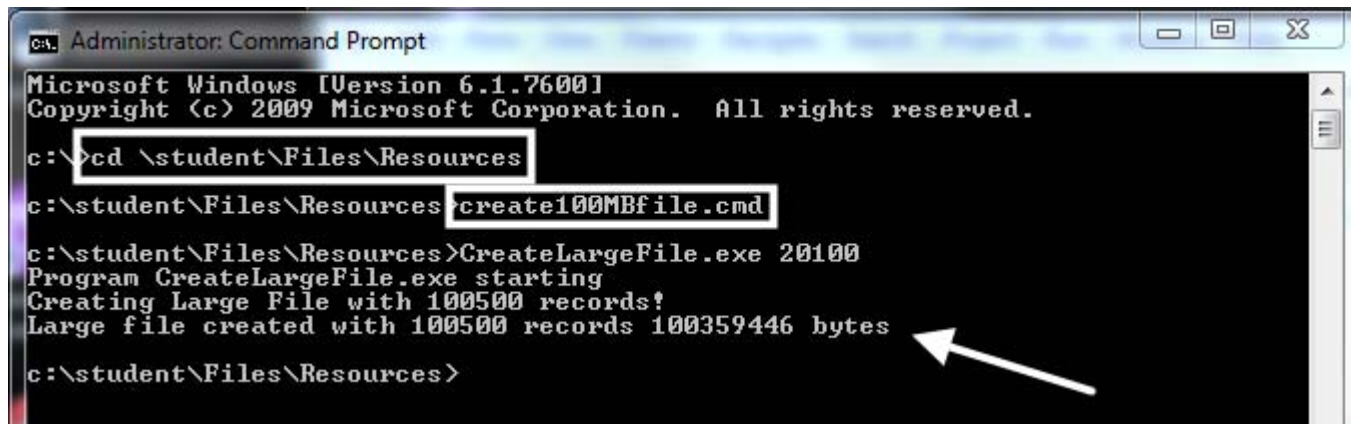
The Windows task manager will also be run during the message flow execution, to show the memory usage as the flow is running.

Start a DOS command prompt, using the icon on the task bar.

- __1. Select the icon for a DOS command prompt in the Windows task bar.
- __2. Press the right mouse button.
- __3. Select **Command Prompt** from the menu.



- __4. Change to the **C:\student\Files\Resources** directory.
- __5. Execute either the **Create100MBfile.cmd** or the **Create1GBfile.cmd** command file.

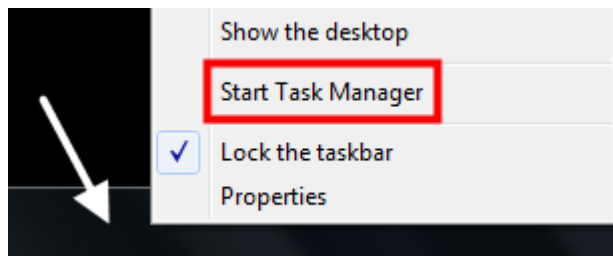


```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\>cd \student\Files\Resources
c:\student\Files\Resources>create100MBfile.cmd
c:\student\Files\Resources>CreateLargeFile.exe 20100
Program CreateLargeFile.exe starting
Creating Large File with 100500 records!
Large file created with 100500 records 100359446 bytes
c:\student\Files\Resources>
```

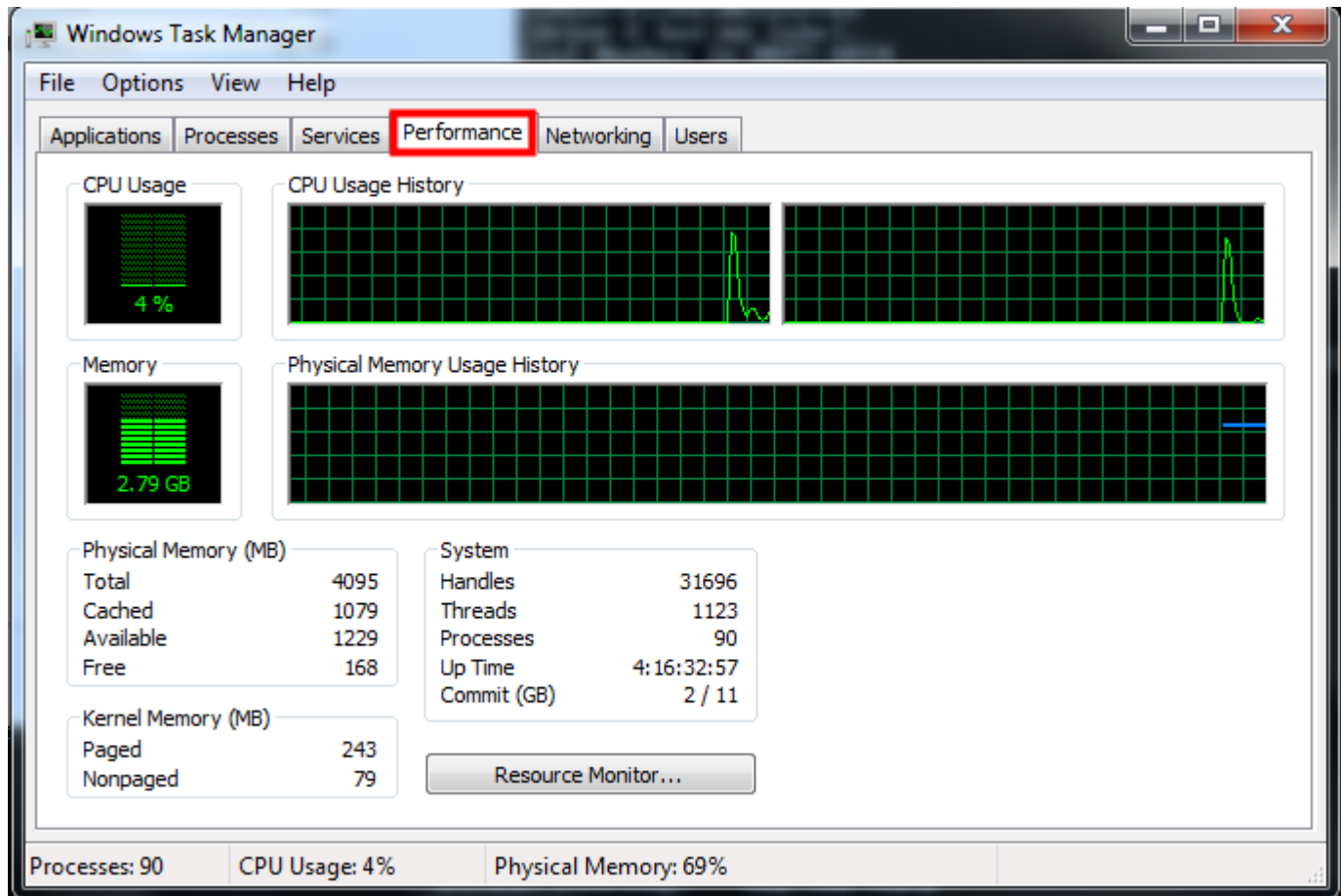
A large file should be created. This will take less than a minute. There is no need to wait for the file to be created.

- __6. Point the mouse cursor to a blank area of the Windows task bar.
- __7. Press the right mouse button.
- __8. Select **Start Task Manager** from the menu.

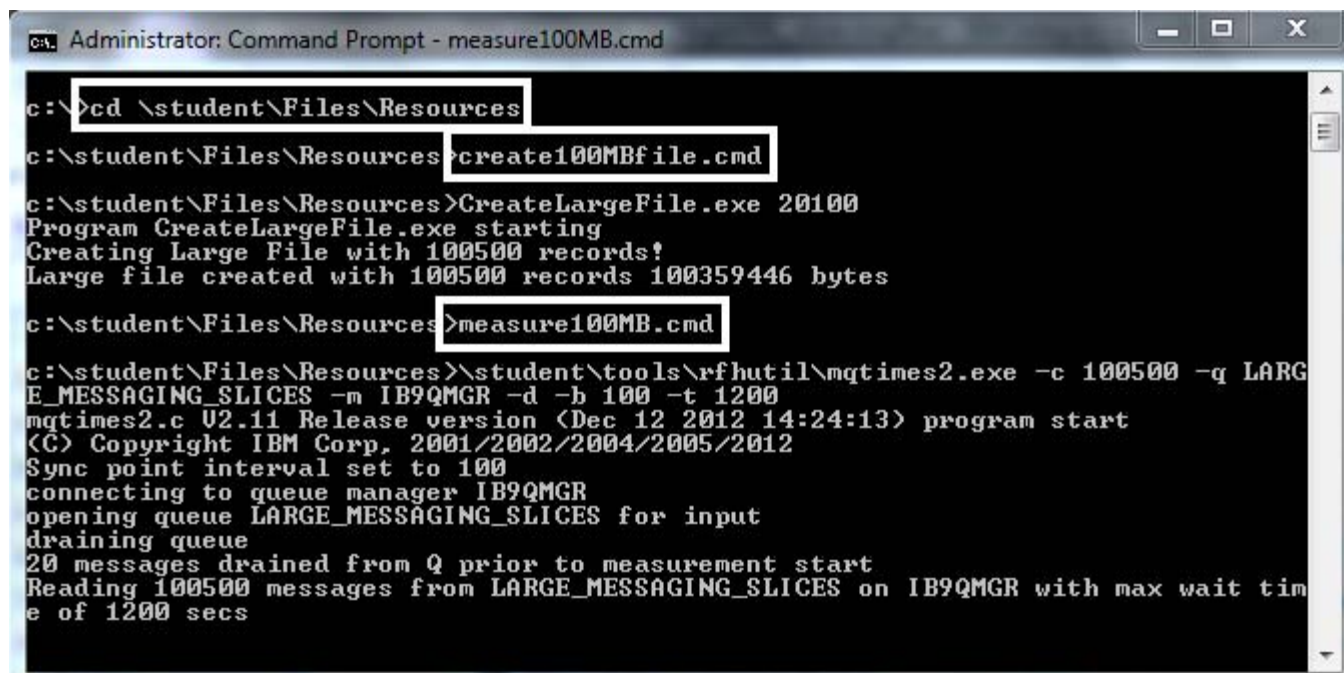


__9. Select the **Performance** tab.

Memory usage should be displayed. Since nothing is running the usage should not be changing.

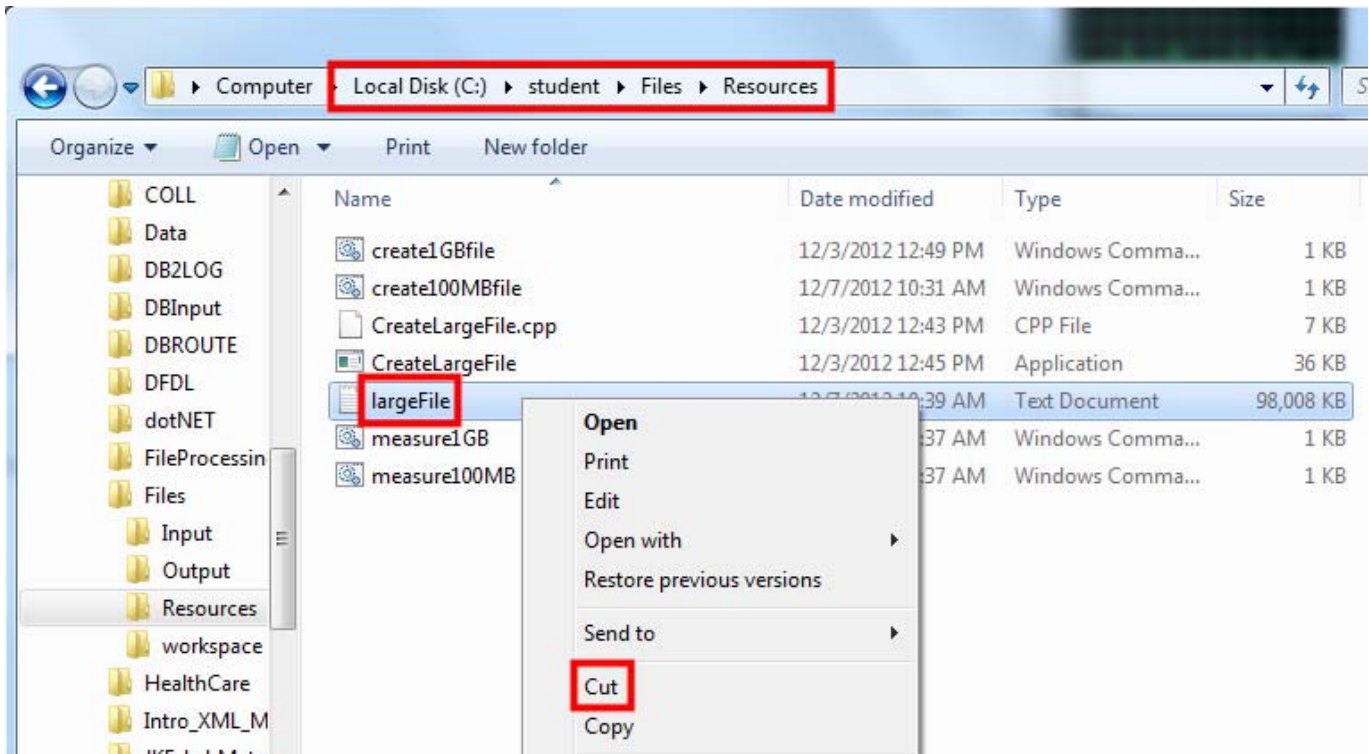


- ___10. Return to the DOS command prompt that was started earlier.
- ___11. Execute the **measure100MB.cmd** or the **measure1GB.cmd** command file, depending on which file was created earlier.

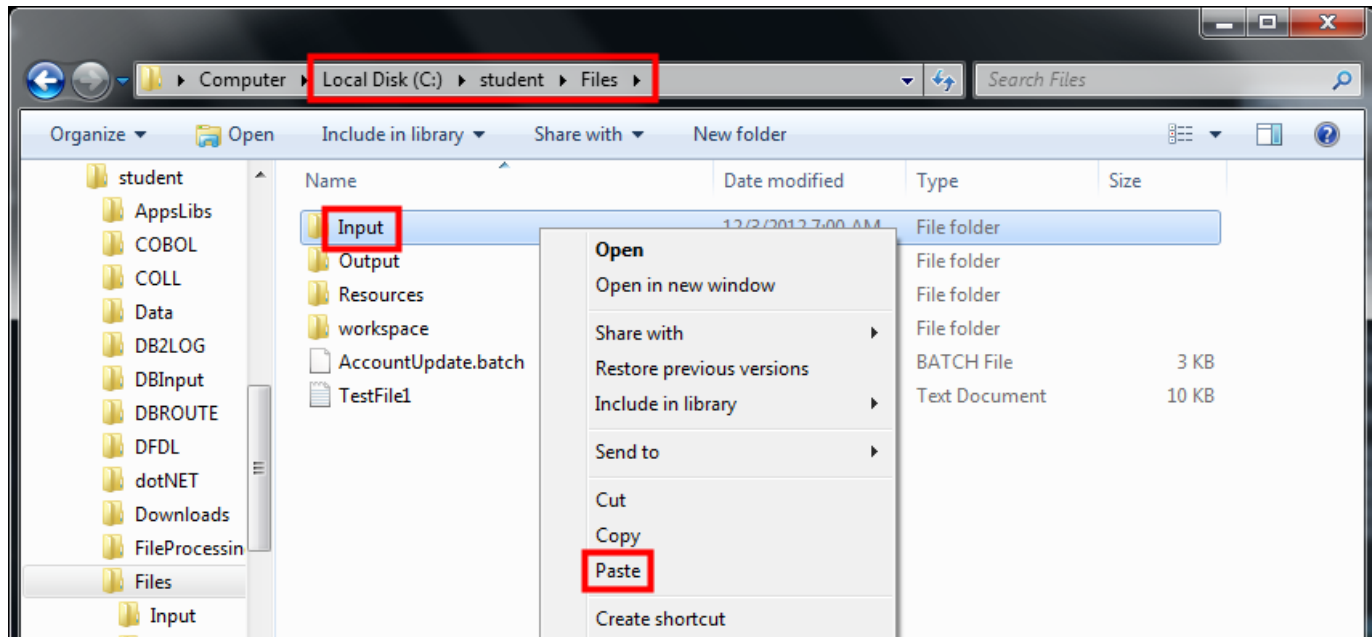


```
c:\>cd \student\Files\Resources
c:\student\Files\Resources>create100MBfile.cmd
c:\student\Files\Resources>CreateLargeFile.exe 20100
Program CreateLargeFile.exe starting
Creating Large File with 100500 records!
Large file created with 100500 records 100359446 bytes
c:\student\Files\Resources>measure100MB.cmd
c:\student\Files\Resources>\student\tools\rfhutil\mqtimes2.exe -c 100500 -q LARG
E_MESSAGING_SLICES -m IB9QMGR -d -b 100 -t 1200
mqtimes2.c V2.11 Release version <Dec 12 2012 14:24:13> program start
(C) Copyright IBM Corp. 2001/2002/2004/2005/2012
Sync point interval set to 100
connecting to queue manager IB9QMGR
opening queue LARGE_MESSAGING_SLICES for input
draining queue
20 messages drained from Q prior to measurement start
Reading 100500 messages from LARGE_MESSAGING_SLICES on IB9QMGR with max wait tim
e of 1200 secs
```

- __12. Return to the Windows Explorer window.
- __13. Navigate to the **C:\student\Files\Resources** directory.
- __14. Select the **largeFile** text file.
- __15. Press the right mouse button.
- __16. Select **Cut** from the menu.



- __17. Navigate to the **C:\student\Files** directory.
- __18. Select the **Input** folder.
- __19. Press the right mouse button.
- __20. Select **Paste** from the menu.



After a brief pause messages should start to be written to the output queue. The number of messages that are written each second is reported.

```

Administrator: Command Prompt - measure100MB.cmd

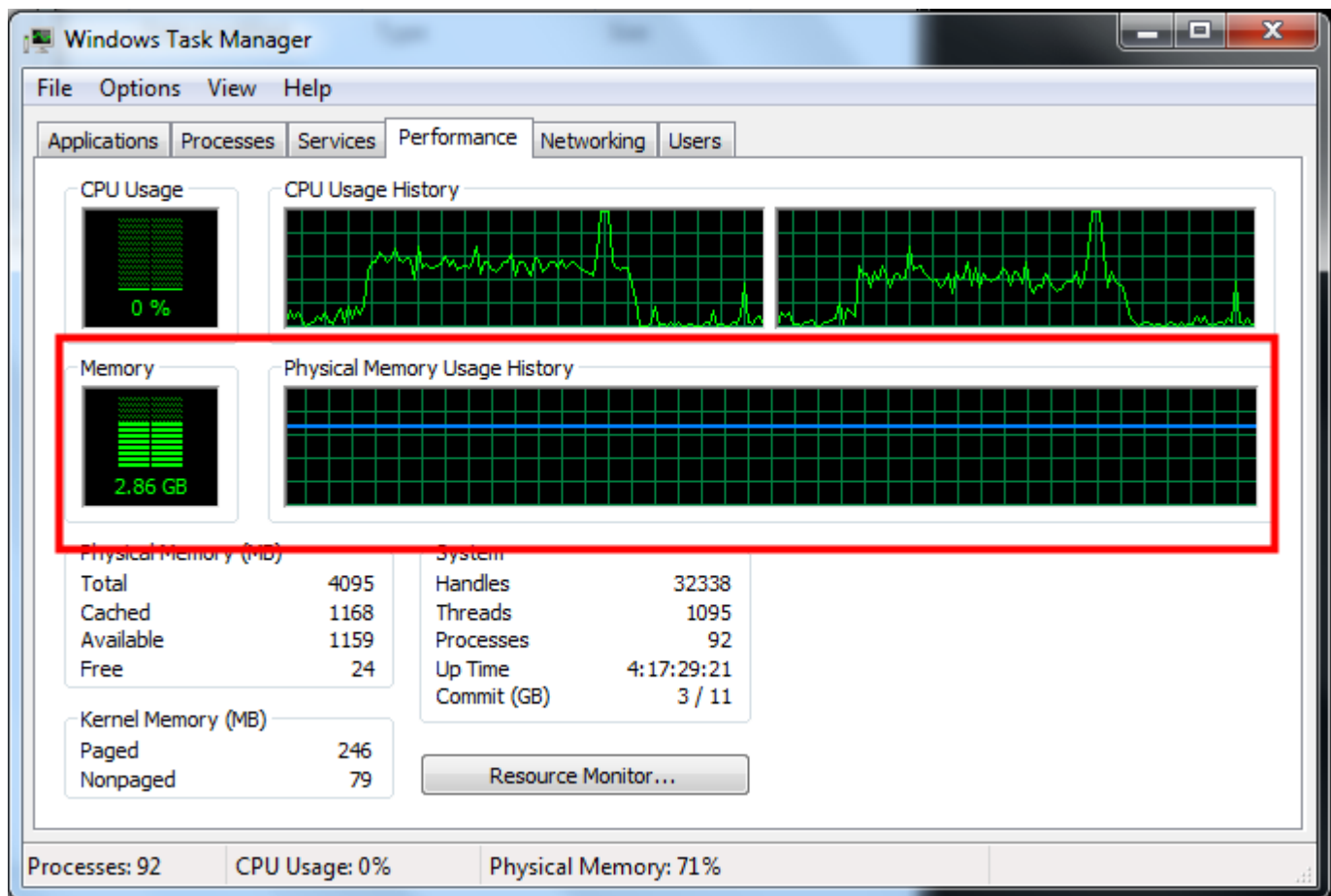
Large file created with 100500 records 100359446 bytes

c:\student\Files\Resources>measure100MB.cmd

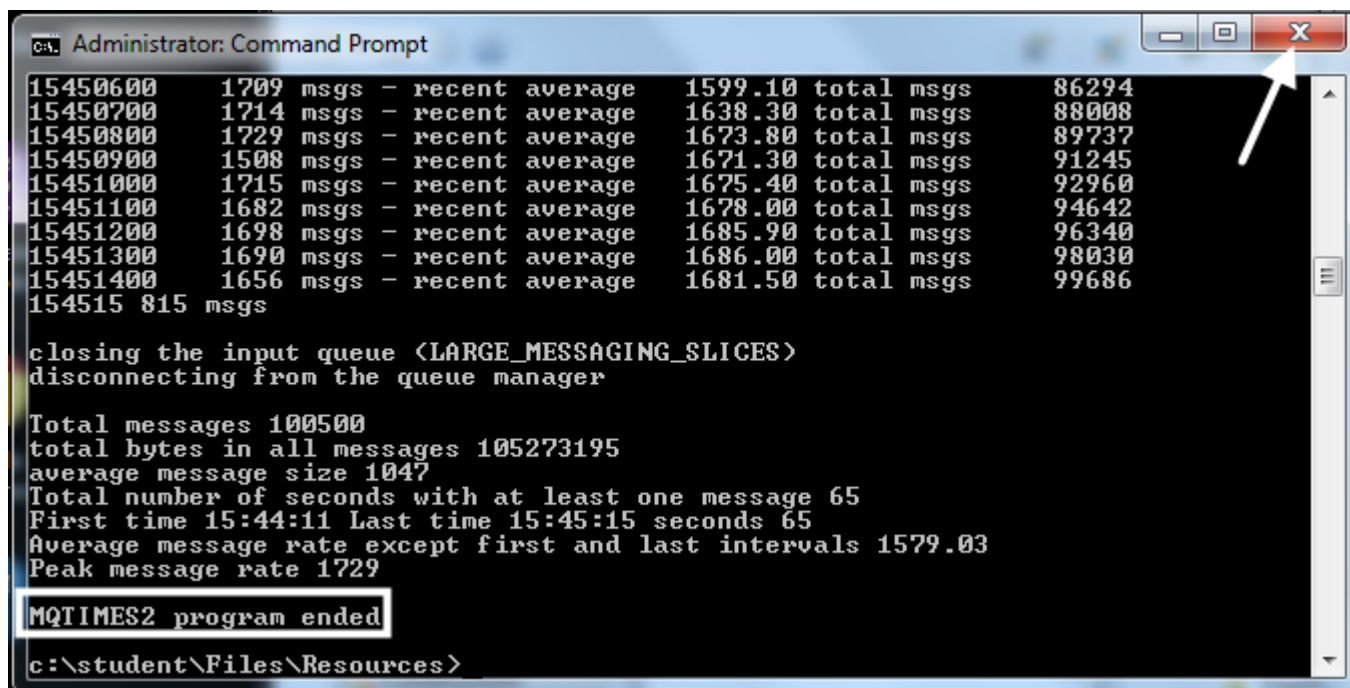
c:\student\Files\Resources>\student\tools\rfhutil\mqtimes2.exe -c 100500 -q LARGE_MESSAGING_SLICES -m IB9QMGR -d -b 100 -t 1200
mqtimes2.c V2.11 Release version (Dec 12 2012 14:24:13) program start
(C) Copyright IBM Corp, 2001/2002/2004/2005/2012
Sync point interval set to 100
connecting to queue manager IB9QMGR
opening queue LARGE_MESSAGING_SLICES for input
draining queue
20 messages drained from Q prior to measurement start
Reading 100500 messages from LARGE_MESSAGING_SLICES on IB9QMGR with max wait time of 1200 secs

21000858      591 msgs total msgs      591
21000900     1583 msgs - recent average 1583.00 total msgs      2174
21001000     1550 msgs - recent average 1566.50 total msgs      3724
21001100     1615 msgs - recent average 1582.67 total msgs      5339
21001200     1432 msgs - recent average 1545.00 total msgs      6771
21001301     1515 msgs - recent average 1539.00 total msgs      8286
21001401     1558 msgs - recent average 1542.17 total msgs     9844
21001501     1617 msgs - recent average 1552.86 total msgs    11461
  
```


Observe the memory usage. Note that the usage is not increasing as the file is being processed.



- __21. When the measurement program is complete close the DOS command prompt window.
- __22. Close the Windows Task Manager.



```

Administrator: Command Prompt

15450600    1709 msgs - recent average 1599.10 total msgs 86294
15450700    1714 msgs - recent average 1638.30 total msgs 88008
15450800    1729 msgs - recent average 1673.80 total msgs 89737
15450900    1508 msgs - recent average 1671.30 total msgs 91245
15451000    1715 msgs - recent average 1675.40 total msgs 92960
15451100    1682 msgs - recent average 1678.00 total msgs 94642
15451200    1698 msgs - recent average 1685.90 total msgs 96340
15451300    1690 msgs - recent average 1686.00 total msgs 98030
15451400    1656 msgs - recent average 1681.50 total msgs 99686
154515 815 msgs

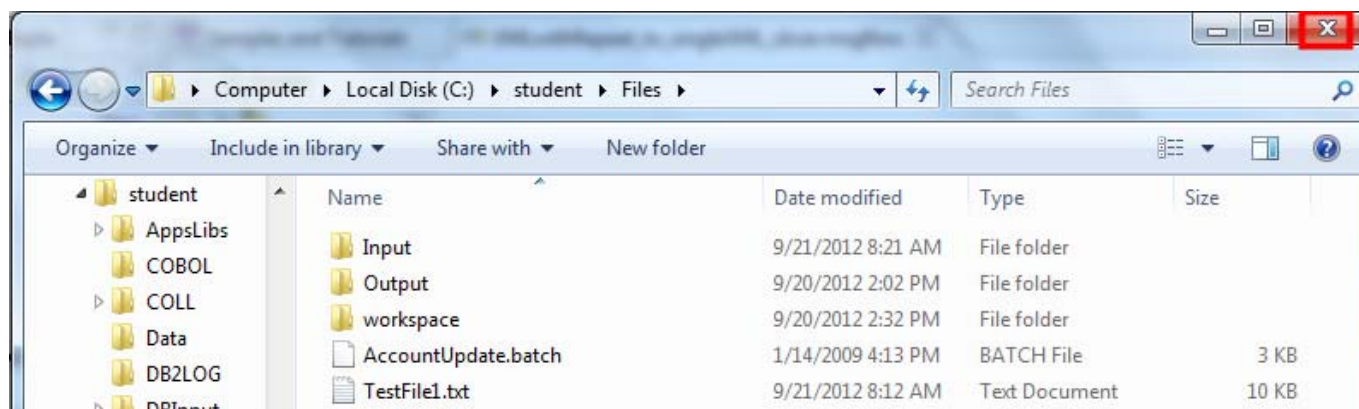
closing the input queue <LARGE_MESSAGING_SLICES>
disconnecting from the queue manager

Total messages 100500
total bytes in all messages 105273195
average message size 1047
Total number of seconds with at least one message 65
First time 15:44:11 Last time 15:45:15 seconds 65
Average message rate except first and last intervals 1579.03
Peak message rate 1729

MQTIMES2 program ended

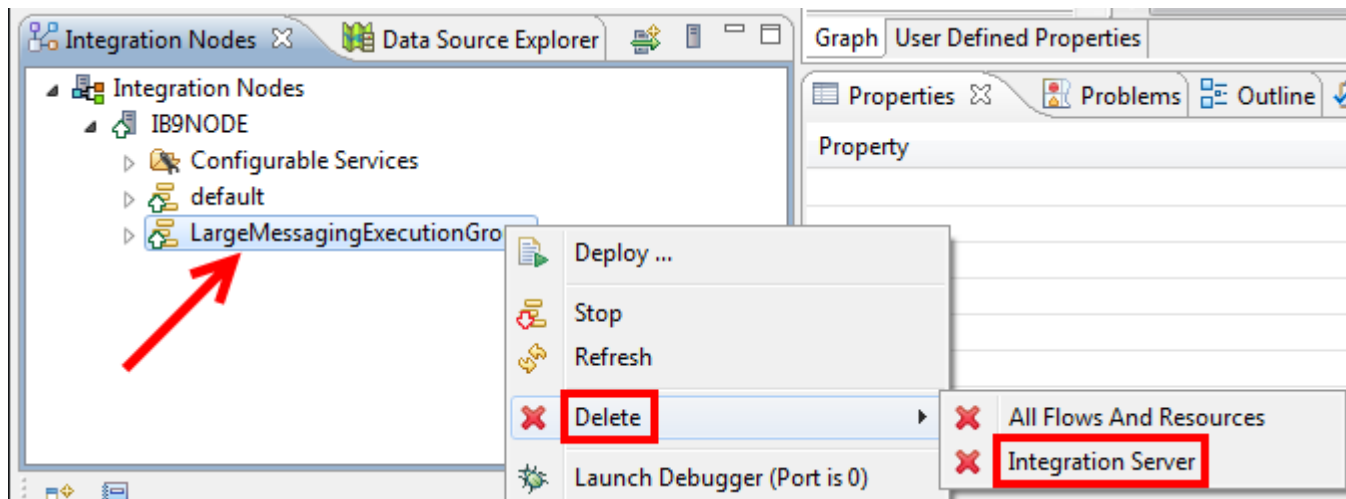
c:\student\Files\Resources>
  
```

- __23. Close the Windows Explorer.

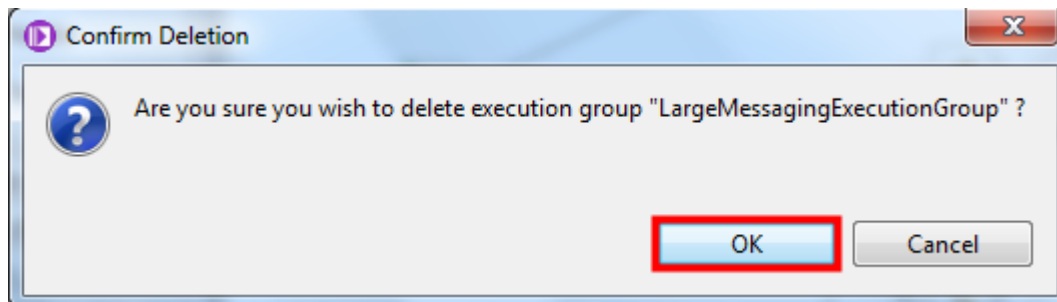


7.5 Clean up

- ___1. In the integration toolkit select the **LargeMessagingExecutionGroup** integration server.
- ___2. Press the right mouse button.
- ___3. Select **Delete→Integration Server** from the menu.



- ___4. Press the **OK** button to confirm the deletion.



This is the end of lab 7.

Lab 8 DFDL and message model tooling

8.1 Introduction to message model standards

A message model is used by IBM Integration Bus to model a message format. The message models are all based on World Wide Web Consortium (W3C) XML Schema 1.0 (XSD).

XML Schema is an international standard that defines a language for describing the structure of XML documents. It is suited to describing the messages that flow between business applications, and it is widely used in the business community for this purpose. IBM Integration Bus uses models that are based on XML Schema to describe the structure of all kinds of message formats, including message formats that are not XML.

Data Format Description Language 1.0 (DFDL) is an open standard modeling language from the Open Grid Forum (OGF) that builds upon the features of XML Schema 1.0 in order to model and validate all kinds of general text and binary data. It uses standard XSD model objects to describe the logical structure of data, together with DFDL annotations that describe the physical text or binary representation of data. IBM Integration Bus uses DFDL schema files to describe text and binary data, including industry standard formats.

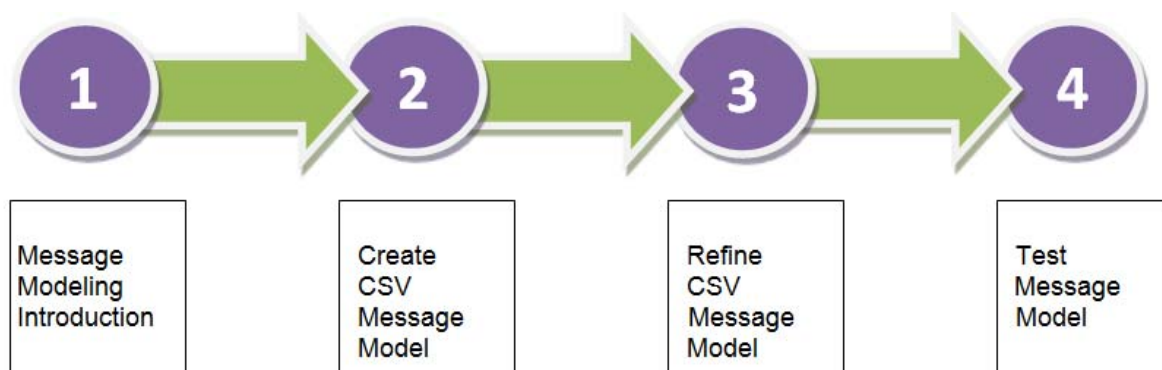
DFDL is not intended to be used to model XML documents (use normal XML Schema files).

Support for DFDL in IBM Integration Bus includes:

- DFDL parser and domain.
- DFDL schema file creation wizards.
- DFDL schema editor for modeling text and binary data formats.
- DFDL Test perspective for testing your DFDL schema files.

For more information about DFDL, you can go to the [Open Grid Forum \(OGF\)](#) website. IBM Integration Bus supports DFDL 1.0, as defined in the following document: [Data Format Description Language \(DFDL\) 1.0](#).

Unlike previous versions of IBM Integration Bus, you don't need to create a Message Set project and a Message Set to model a message type (although MRM Message Models are still supported); you just have to create a DFDL schema file.



8.2 Creating a CSV message model

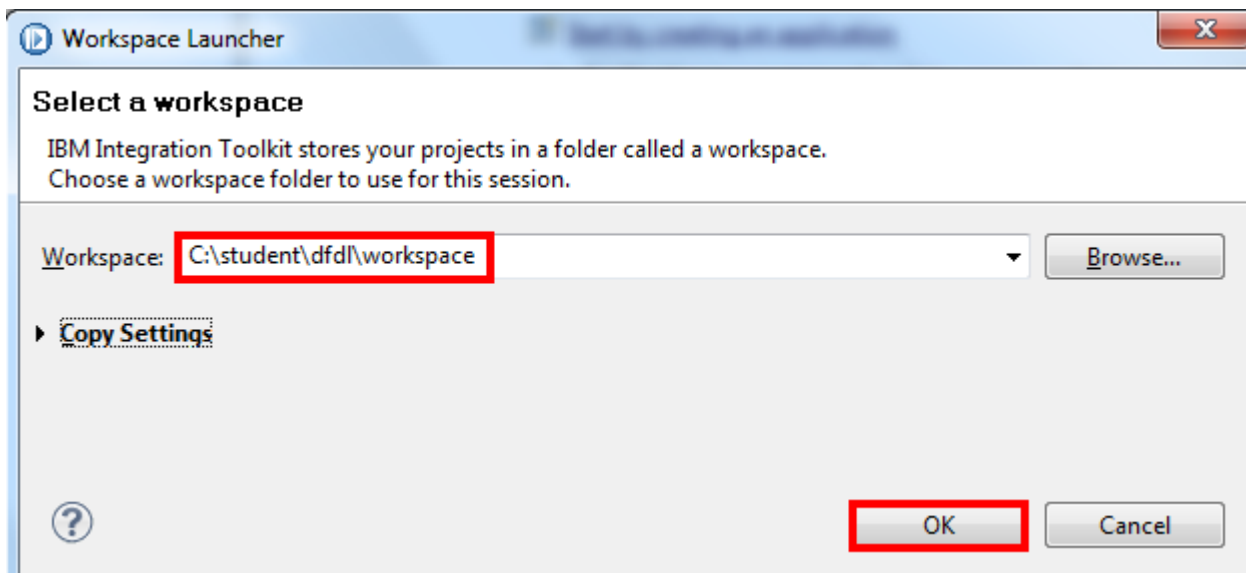
This lab will create a message model that will model a CSV file. The file has a header record, and several detail records, but no trailer record.

Records are delimited by CRLF characters, and fields within each record are delimited by a comma.

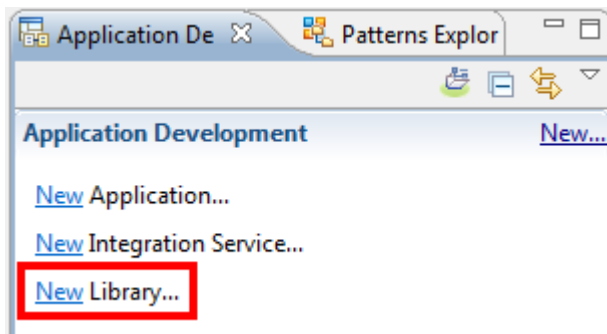
The Library project type will be used to store the message model.

This lab will start from a blank workspace.

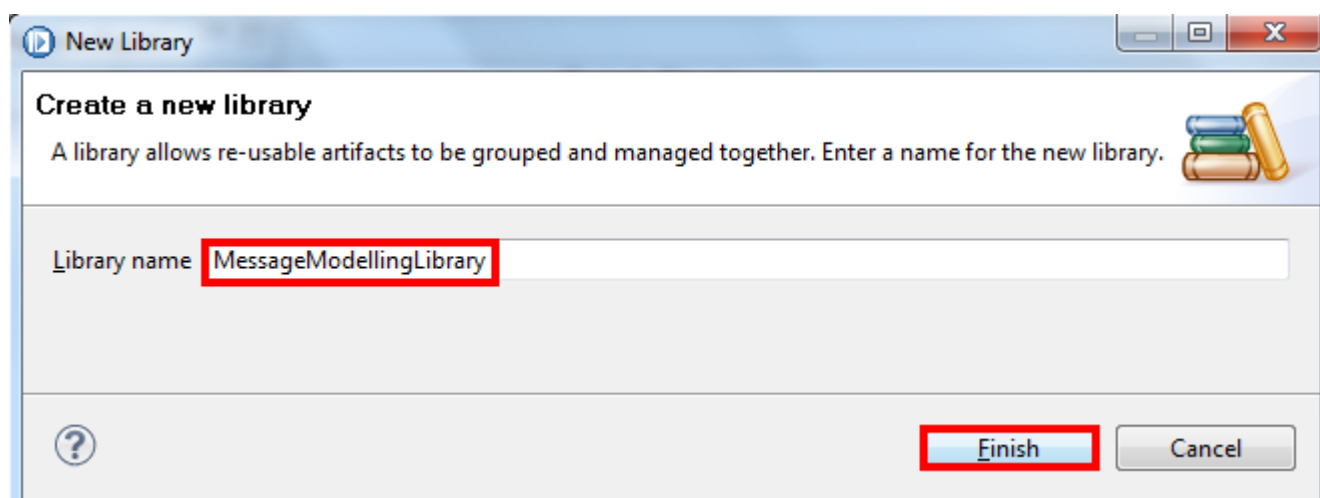
- __1. Select **File→Switch Workspace→Other**.
- __2. Use the **Browse** button to select the **C:\student\DFDL\workspace** workspace.
- __3. Press the **OK** button to continue.



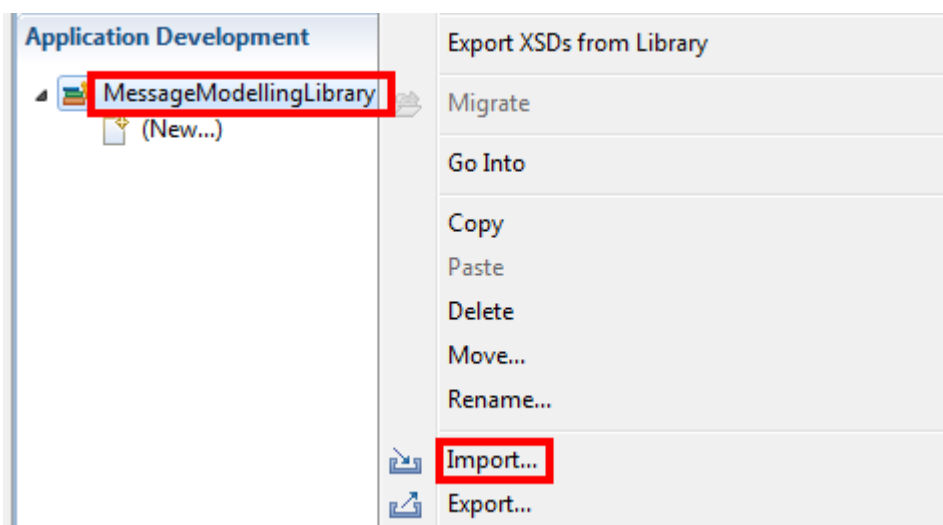
- __4. Click **New Library**.



- __5. Set the **Library name** to **MessageModellingLibrary**.
- __6. Press the **Finish** button.



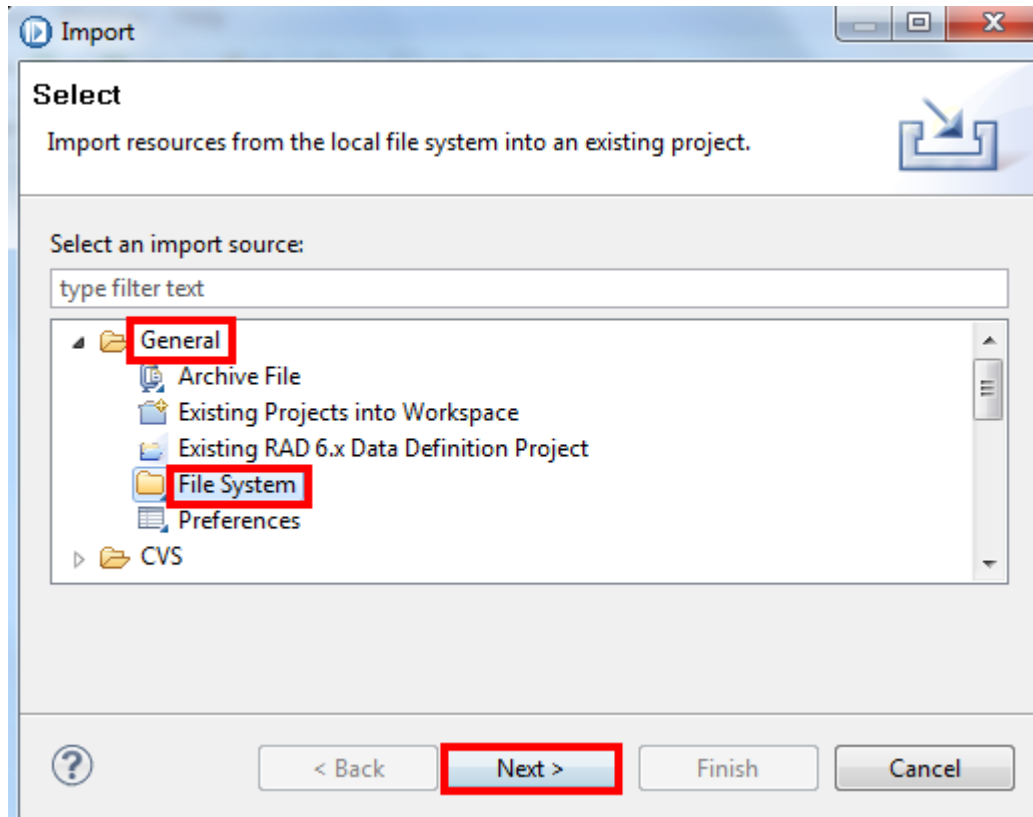
- __7. Select the **MessageModellingLibrary** library.
- __8. Press the right mouse button.
- __9. Select **Import** from the menu



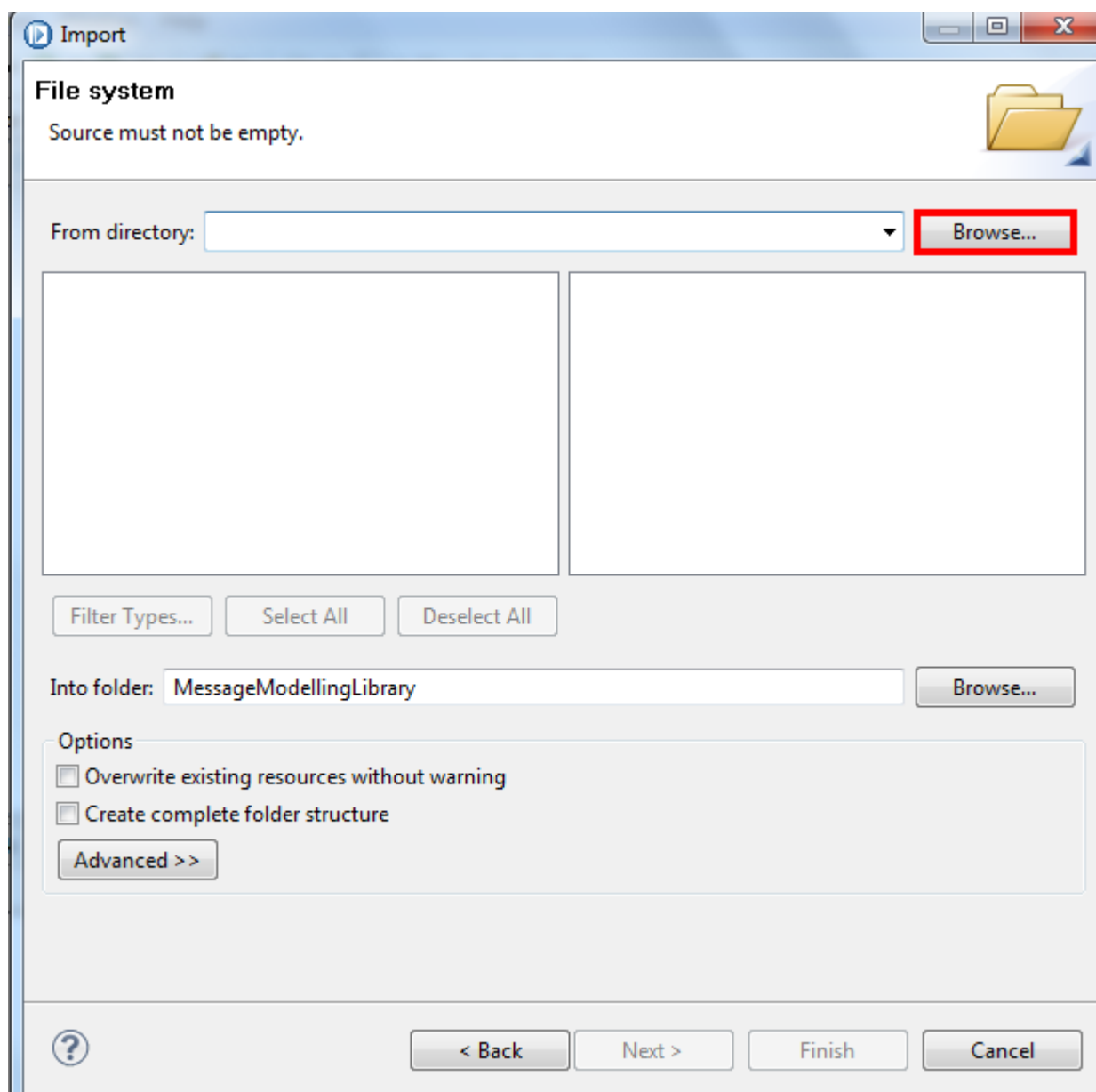
__10. Expand the **General** folder.

__11. Select **File System**.

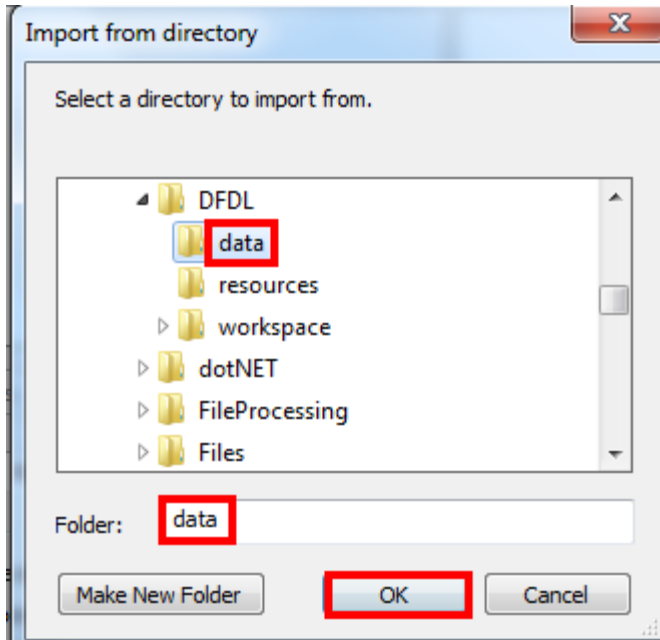
__12. Press the **Next** button.



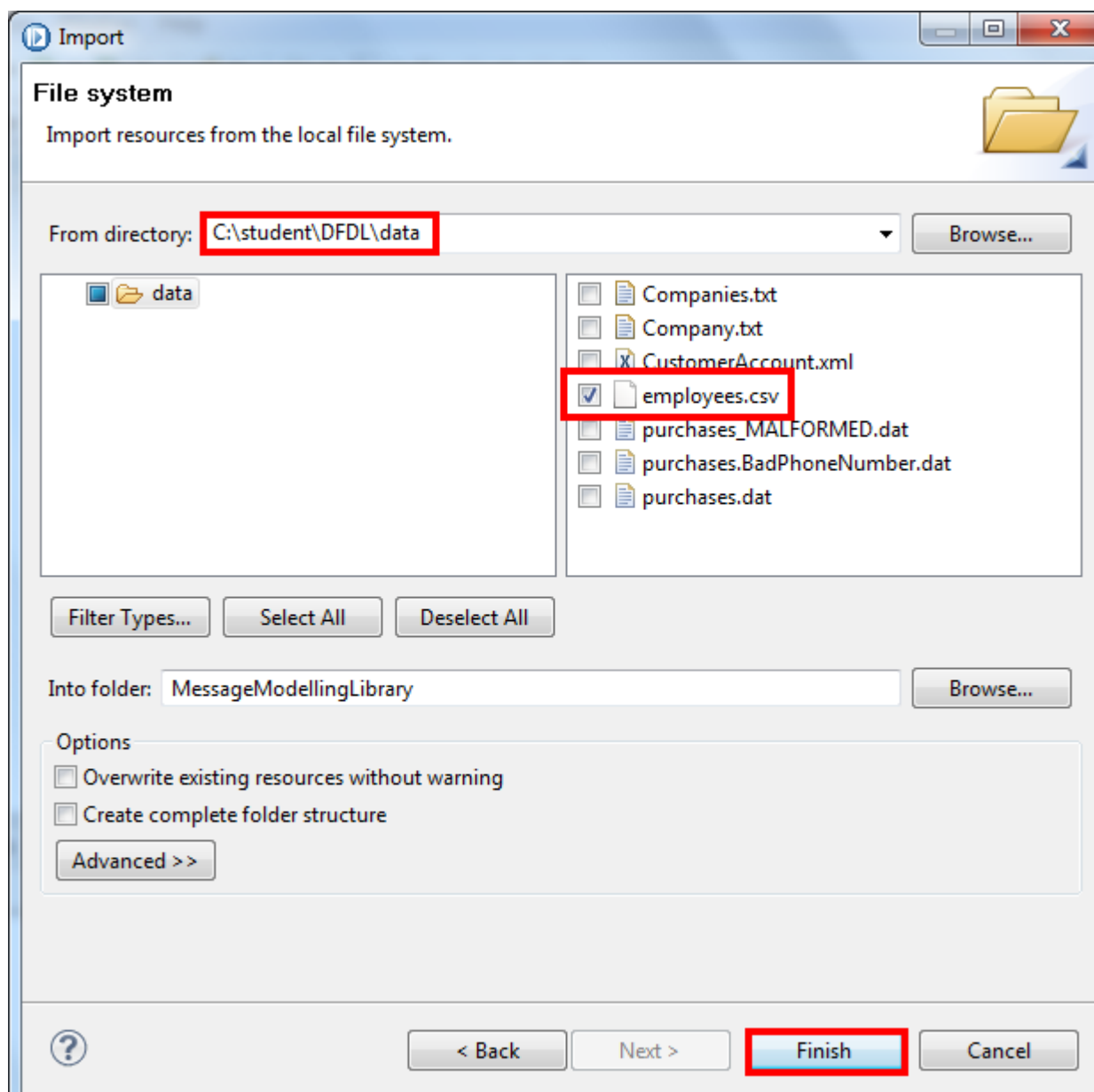
__13. Press the **Browse** button for **From directory**.



- __14. Navigate to the **C:\student\DFDL\data** directory.
- __15. Select the **data** directory.
- __16. Press the **OK** button.

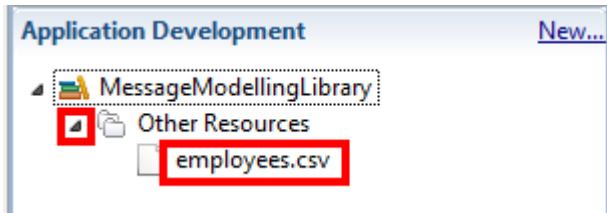


- __17. Select the **employees.csv** file.
- __18. Press the **Finish** button to perform the import.



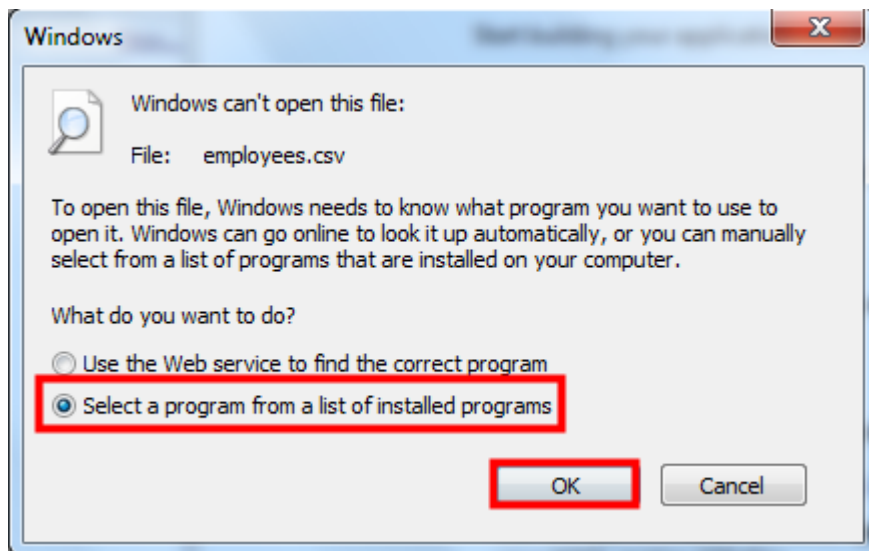
__19. Expand the **Other Resources** folder.

__20. Double-click the **employees.csv** file.



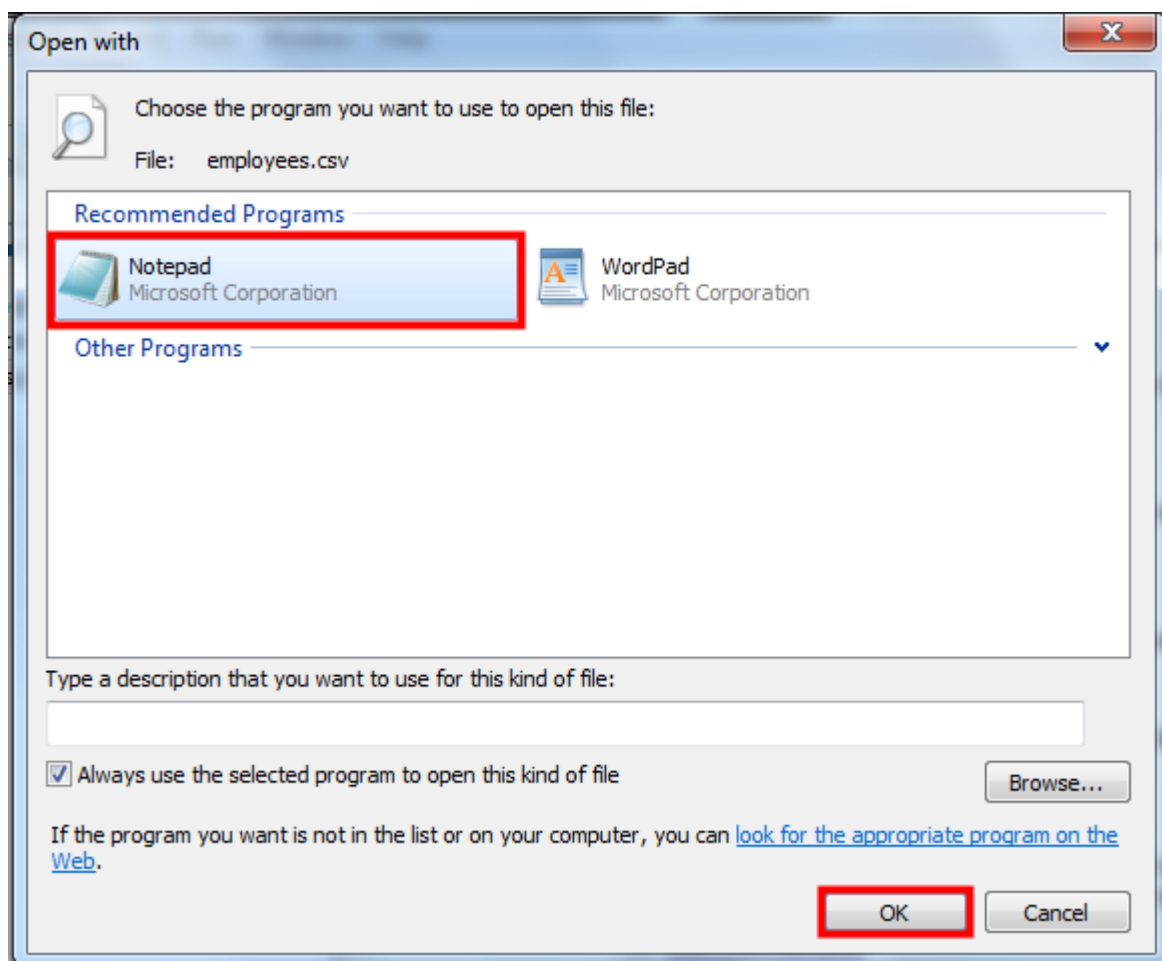
__21. Select the **Select a program from a list of installed programs** radio button.

__22. Press the **OK** button.



__23. Select the **Notepad** utility.

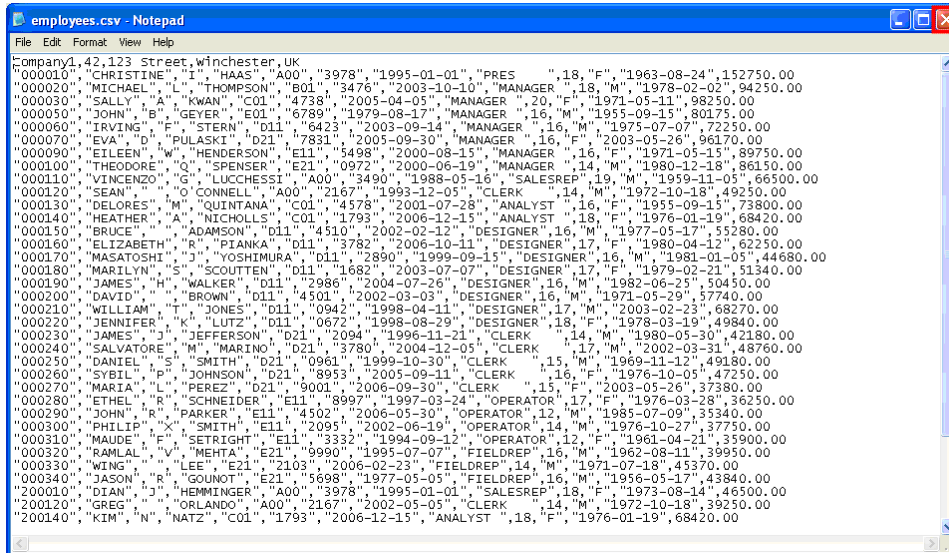
__24. Press the **OK** button.



The file data will be opened in a notepad session. There is a header record and 42 detail records. Each detail record has 12 fields.

__25. Examine the file data.

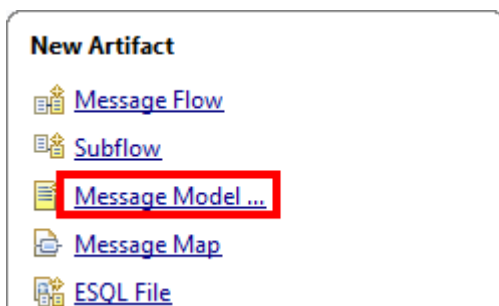
__26. Close the notepad window.



__27. In the Integration Toolkit click **New**.



__28. Select **Message Model** from the menu.



__29. Select the **CSV text** radio button.

__30. Press the **Next** button.

New Message Model

Create a new message model file

Select the message model type or format

XML

- ☐ **SOAP XML** XML data for use in Web Services.
- ☐ **Other XML** All other XML data.

Text and binary

- ☒ **CSV text** Comma Separated Values data, a delimited text format commonly used as an export format by spreadsheets and databases.
- ☐ **Record-oriented text** Text data formats where delimited fields are grouped into records.
- ☐ **COBOL** Data for COBOL programs
- ☐ **C** Data for C programs
- ☐ **Other text or binary** All other text or binary data formats.

Enterprise Information Systems

- ☐ **SAP** Data from SAP systems including IDoc and BAPI
- ☐ **Siebel** Data from Siebel systems
- ☐ **PeopleSoft** Data from PeopleSoft
- ☐ **JD Edwards** Data from JD Edwards systems

Other

- ☐ **CORBA IDL** Data from CORBA
- ☐ **Database record** Records from relational databases
- ☐ **MIME** Data for extended email format
- ☐ **IBM supplied** Predefined data format

Navigation: ? < Back **Next >** Finish Cancel

- __31. Select the **Create a DFDL schema file using the wizard to guide you** radio button.
- __32. Press the **Next** button.

New Message Model

CSV text

Choose how you would like to create your CSV message model.

WebSphere Message Broker requires a message model in order to parse, serialize and validate CSV data. A message model also speeds up development of your message broker applications by enabling ESQL content assist and graphical maps.

Comma Separated Values data is modeled by Data Format Description Language (DFDL) schema files. DFDL is a standard from the Open Grid Forum for describing all kinds of text and binary data.

☒ Create a DFDL schema file using this wizard to guide you.

☐ Create an empty DFDL schema file, I will model my data using the DFDL schema editor

☐ Import or replace the IBM supplied DFDL schema property defaults for CSV.

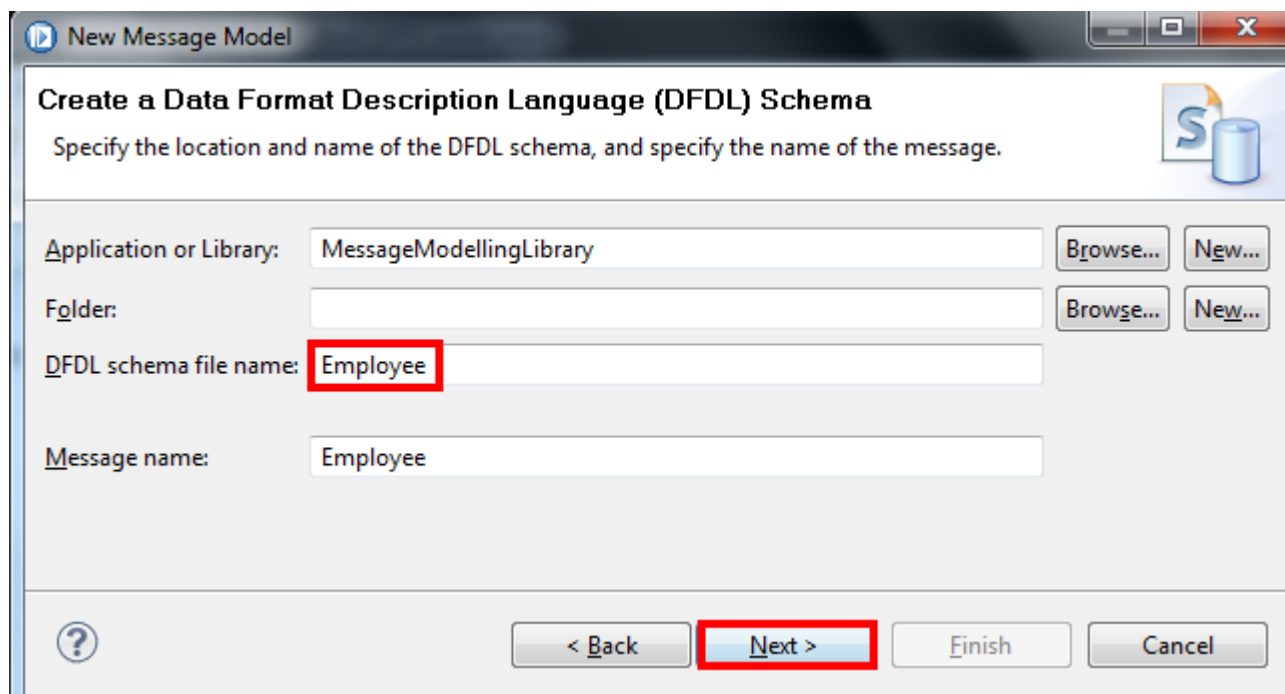
	A	B	C	D	E
1	Year	Make	Model	Description	Price
2	2009	SK Inc	MBTk7	4293cc, V8	53880.00
3	2010	Hans On	DFDL	3000cc straight 6	31395.00
4	2010	AOD corp	MB8	4163cc, V8	51435.00

Export

Year,Make,Model,Description,Price
2009,SK Inc,MBTk7,"4293cc, V8",53880.00
2010,Hans On,DFDL,3000cc straight 6,31395.00
2010,AOD corp,MB8,"4163cc, V8",51435.00

< Back **Next >** Finish Cancel

- __33. Make sure that the **Project** is set to **MessageModellingLibrary**.
- __34. Enter **Employee** as the **DFDL schema file name**.
- __35. Press the **Next** button.



New Message Model

Create a Data Format Description Language (DFDL) Schema

Specify the location and name of the DFDL schema, and specify the name of the message.

Application or Library: MessageModellingLibrary **Browse...** **New...**

Folder: **Browse...** **New...**

DFDL schema file name: Employee

Message name: Employee

< Back **Next >** **Finish** **Cancel**

__36. Select the **first record is a header** check box.

__37. Enter **12** as the **Number of fields**.

__38. Press the **Finish** button.

New Message Model

Configure schema for CSV data
Provide settings for a new schema that will model CSV data.

Record settings
End of record character: Carriage Return & Line Feed - %CR;%LF;
(Blank records will be skipped)
☒ The first record is a header

Field settings
Number of fields: 12
☐ Create default values for fields

Encoding code page options:
☒ Dynamic (provided to the processor by the application at runtime)
☐ Fixed UTF-8

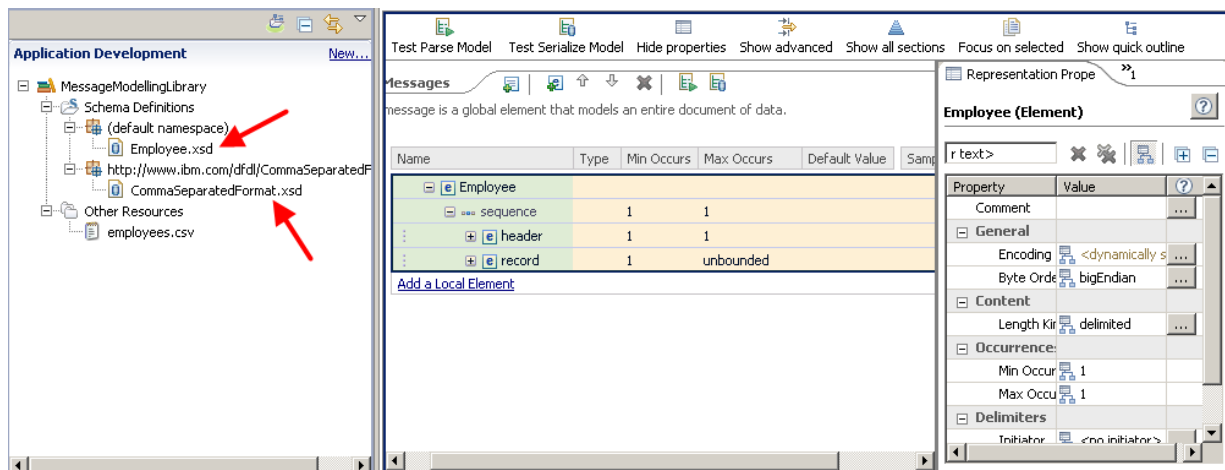
Global settings
Escape scheme: CSV Escape Scheme

? < Back Next > **Finish** Cancel

There should be a new folder in the library with an XSD file called **Employee.xsd** (under the **default namespace**). This is the message model that was just created. This is a standard DFDL XSD, with no specific annotations for IBM Integration Bus.

There is another XSD file under Schema Definitions called "CommaSeparatedFormat.xsd". This "Helper schema file" was automatically added by the CSV wizard and contains CSV-specific defaults for all the DFDL properties. This is required because DFDL doesn't have built-in defaults, so if an object needs a property, a value must be supplied. To ease this task, the wizard creates a helper DFDL schema for each kind of data (COBOL, CSV and others).

These files are related by an import statement in the schema references section of the **Employee.xsd** file.



Take a look at the Representation Properties view on the right-hand side. Notice the "inheritance" icon to the right of the "**Length Kind**" property. (**Hint:** You might have to adjust the column lengths in the Representation Properties view to see the full contents.)

In this case, the "**Length Kind**" property was inherited from the "**CommaSeparatedFormat**" helper schema file that the wizard automatically imported.

__39. Hover over the inheritance icon to discover its origin.

The screenshot shows the IBM Data Architect interface. On the left, the 'Messages' view displays a table for the 'Employee' element:

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		

On the right, the 'Representation Properties' view for the 'Employee (Element)' is shown. It has a search bar 'type filter text>' and a table of properties:

Property	Value
Comment	
General	
Encoding (code)	< dynamically s

A red arrow points to the inheritance icon (a small square with a diagonal line) next to the 'Encoding (code)' property. A tooltip is displayed over this icon with the following text:

Property inherited from global named format
<http://www.ibm.com/dfdl/CommaSeparatedFormat/CommaSeparatedFormat>

At the bottom right of the tooltip, it says 'Press "F2" for focus'.

The DFDL Editor in the center of the screen describes data elements. Expand the header element.

Take a look at the columns:

- Name: Name of the data element
- Type: Data type (string, int, Boolean, decimal, date and so on)
- Min Occurs: Minimum amount of occurrences expected (0 or greater)
- Max Occurs: Maximum amount of occurrences expected (from 1 to unbounded)
- Default Value: Used to provide the logical value of a required element while parsing or serializing messages when the element is missing
- Sample Test Data: Can be used to generate a logical instance of the model as you will see later on

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
sequence		1	1		
head_field1	string	1	1		head_value1
head_field2	string	1	1		head_value2
head_field3	string	1	1		head_value3
head_field4	string	1	1		head_value4
head_field5	string	1	1		head_value5
head_field6	string	1	1		head_value6
head_field7	string	1	1		head_value7
head_field8	string	1	1		head_value8
head_field9	string	1	1		head_value9
head_field10	string	1	1		head_value10
head_field11	string	1	1		head_value11
head_field12	string	1	1		head_value12
record		1	unbounded		
Add a Local Element					

Fields 6 to 12 will now be deleted from the **header** element.

__40. Expand the **header** element.

__41. Select **head_field6**.

__42. Hold down the **Ctrl** key and select fields **head_field7** through **head_field12**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
sequence		1	1		
head_field1	string	1	1		head_value1
head_field2	string	1	1		head_value2
head_field3	string	1	1		head_value3
head_field4	string	1	1		head_value4
head_field5	string	1	1		head_value5
head_field6	string	1	1		head_value6
head_field7	string	1	1		head_value7
head_field8	string	1	1		head_value8
head_field9	string	1	1		head_value9
head_field10	string	1	1		head_value10
head_field11	string	1	1		head_value11
head_field12	string	1	1		head_value12
record		1	unbounded		
Add a Local Element					

__43. Press the right mouse button.

__44. Select **Delete**.

N.B. You can also just press the **Delete** key.

				1	head_value6
				1	head_value7
				1	head_value8
				1	head_value9
				1	head_value10
				1	head_value11
				1	head_value12
				1	unbounded
Add a Local Element					

Change the names of the remaining five header fields as shown.

__45. Click the name of the first element.

__46. Change the name as shown.

__47. Use the down arrow key to move to the next element name. Press the **Enter** key when the last name is changed.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
sequence		1	1		
CompanyName	string	1	1		head_value1
EmpCount	string	1	1		head_value2
Address	string	1	1		head_value3
City	string	1	1		head_value4
Country	string	1	1		head_value5
record		1	unbounded		
Add a Local Element					

__48. Collapse the **header** element.

__49. Expand the **record** element to show the 12 fields created by the wizard.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
... sequence		1	1		
⋮ header		1	1		
⋮ record		1	unbounded		
... sequence		1	1		
⋮ field1	string	1	1		value1
⋮ field2	string	1	1		value2
⋮ field3	string	1	1		value3
⋮ field4	string	1	1		value4
⋮ field5	string	1	1		value5
⋮ field6	string	1	1		value6
⋮ field7	string	1	1		value7
⋮ field8	string	1	1		value8
⋮ field9	string	1	1		value9
⋮ field10	string	1	1		value10
⋮ field11	string	1	1		value11
⋮ field12	string	1	1		value12
Add a Local Element					

__50. Change the field names to the following:

- | | |
|--------------|---------------|
| 1) EmpNo | 7) HireDate |
| 2) FirstName | 8) Job |
| 3) MidInit | 9) EdLevel |
| 4) LastName | 10) Sex |
| 5) WorkDept | 11) BirthDate |
| 6) PhoneNo | 12) Salary |

Again use the down arrow to move from one field to the next. Press the **Enter** key to complete the rename operation.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
[-] Employee					
[-] sequence					
[+] header					
[-] record					
[-] sequence					
[-] EmpNo	string	1	1		value1
[-] FirstName	string	1	1		value2
[-] MidInit	string	1	1		value3
[-] LastName	string	1	1		value4
[-] WorkDept	string	1	1		value5
[-] PhoneNo	string	1	1		value6
[-] HireDate	string	1	1		value7
[-] Job	string	1	1		value8
[-] EdLevel	string	1	1		value9
[-] Sex	string	1	1		value10
[-] BirthDate	string	1	1		value11
[-] Salary	string	1	1		value12
Add a Local Element					

__51. Click in the **Type** column for the **HireDate** field.

__52. Select **date** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
EmpNo	string	1	1		value1
FirstName	string	1	1		value2
MidInit	string	1	1		value3
LastName	string	1	1		value4
WorkDept	string	1	1		value5
PhoneNo	string	1	1		value6
HireDate	string	1	1		value7
Job					value8
EdLevel					value9
Sex					value10
BirthDate					value11
Salary					value12

[Add a Local Element](#)

Browse...

<Anonymous>

- boolean
- byte
- date**
- dateTime
- decimal

__53. Click in the **Type** column for the **BirthDate** field.

__54. Select **date** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
EmpNo					value1
FirstName					value2
MidInit					value3
LastName					value4
WorkDept					value5
PhoneNo					value6
HireDate					2010-12-31
Job					value8
EdLevel					value9
Sex					value10
BirthDate	string	1	1		value11
Salary	string	1	1		value12
Add a Local Element					

Browse...

<Anonymous>

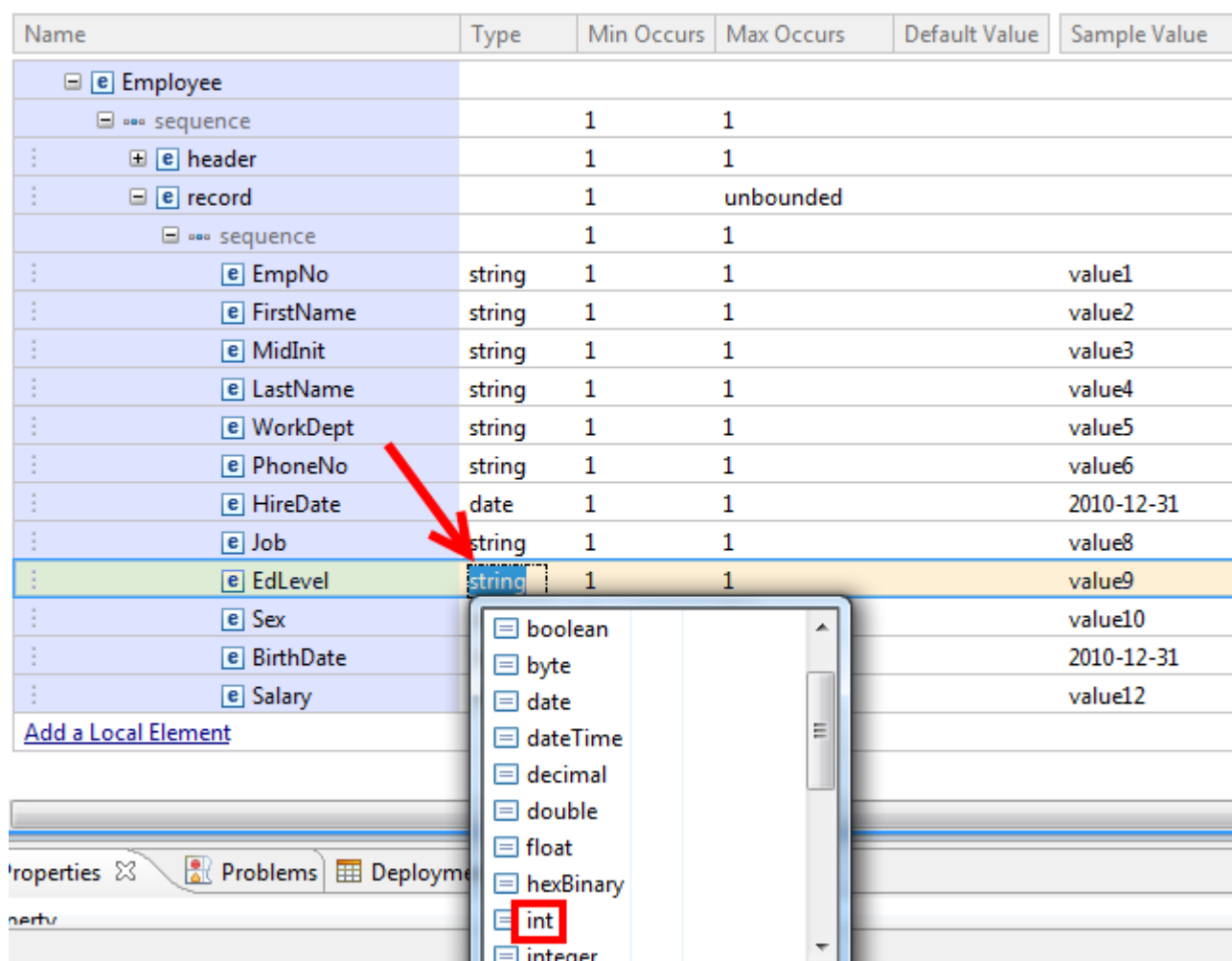
- boolean
- byte
- date**
- dateTime
- decimal
- double
- float
- hexBinary

__55. Click in the **Type** column for the **EdLevel** field.

__56. Select **int** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
EmpNo	string	1	1		value1
FirstName	string	1	1		value2
MidInit	string	1	1		value3
LastName	string	1	1		value4
WorkDept	string	1	1		value5
PhoneNo	string	1	1		value6
HireDate	date	1	1		2010-12-31
Job	string	1	1		value8
EdLevel	string	1	1		value9
Sex					value10
BirthDate					2010-12-31
Salary					value12

[Add a Local Element](#)



- boolean
- byte
- date
- dateTime
- decimal
- double
- float
- hexBinary
- int**
- integer

__57. Click in the **Type** column for the **Salary** field.

__58. Select **decimal** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
[-] [e] Employee					
[-] ... sequence		1	1		
... [+] [e] header		1	1		
... [-] [e] record		1	unbounded		
[-] ... sequence		1	1		
... [e] EmpNo	string	1	1		value1
... [e] FirstName					value2
... [e] MidInit					value3
... [e] LastName					value4
... [e] WorkDept					value5
... [e] PhoneNo					value6
... [e] HireDate					2010-12-31
... [e] Job					value8
... [e] EdLevel					1
... [e] Sex					value10
... [e] BirthDate					2010-12-31
... [e] Salary	string	1	1		value12
Add a Local Element					

Browse...

<Anonymous>

- boolean
- byte
- date
- dateTime
- decimal**
- double
- float
- hexBinary

__60. Select the **record** element (if necessary).

__61. Examine the **Terminator** property.

The screenshot shows the 'Employee.xsd' message model tooling interface. The left pane displays a tree view of the message model. The 'record' element is selected and highlighted with a red box. The right pane shows the 'record (Element)' properties. The 'Terminator' property is highlighted with a red arrow, and its value is '%CR;%LF;%WSP*;'. The 'Delimiters' property is also highlighted with a red arrow.

Name	Type	Min Occurs	Max Occurs	Default
Employee				
sequence		1	1	
header		1	1	
record		1	unbounded	
sequence		1	1	
EmpNo	string	1	1	
FirstName	string	1	1	
MidInit	string	1	1	
LastName	string	1	1	
WorkDept	string	1	1	
PhoneNo	string	1	1	
HireDate	date	1	1	
Job	string	1	1	
EdLevel	int	1	1	

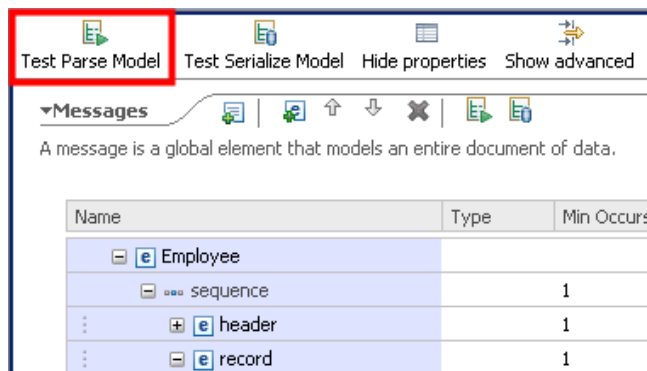
Property	Value
Comment	
General	
Content	
Occurrences	
Max Occurs	unbounded
Occurs Count Kind	implicit
Delimiters	
Terminator	%CR;%LF;%WSP*;

__62. Save the message model (**Ctrl+S**).

8.3 Test the message model

The message model will be tested to validate that it parses the sample data correctly.

___1. Click the **Test Parse Model** button.



- __2. Select the **Content from a data file** radio button.
- __3. Press the **Browse** button.

Test Parse Model

Message
Select message for testing. [More...](#)
Message name:* Employee

Parser Input
Select content to be parsed against schema.
☐ Content from 'DFDL Test - Serialize' view
☒ Content from a data file
Input file name:* **Browse...**

Specify runtime configuration.

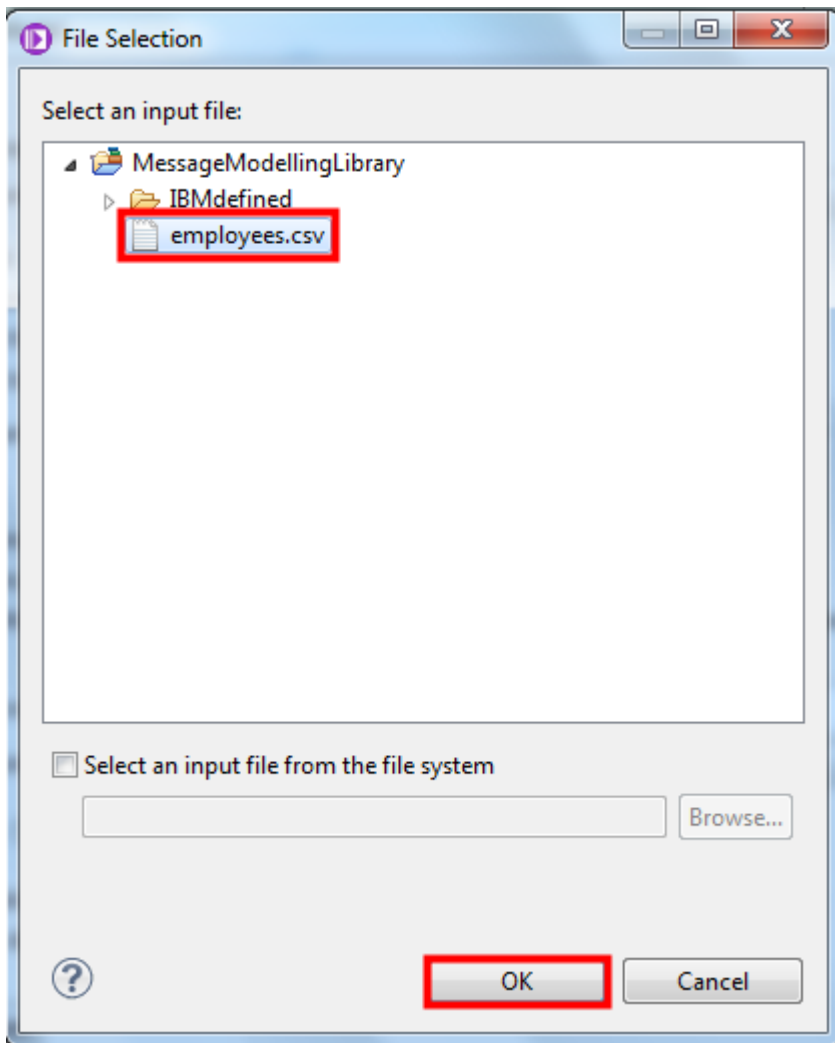
Runtime encoding options
Provide runtime values for properties which have been configured in the model to be dynamically set. [More...](#)
Encoding (code page): UTF-8
Floating point format: IEEE Non-Extended
Byte order: ☐ Little endian ☒ Big endian

Runtime validation
☐ Validate data against schema [More...](#)

Restore Defaults

OK Cancel

- __4. Select the **employees.csv** file under the **MessageModellingLibrary** library.
- __5. Press the **OK** button.



__6. Press the **OK** button.

Test Parse Model

Message
Select message for testing. [More...](#)
Message name:* Employee

Parser Input
Select content to be parsed against schema.
☐ Content from 'DFDL Test - Serialize' view
☒ Content from a data file
Input file name:* /MessageModellingLibrary/employees.csv [Browse...](#)

Specify runtime configuration.

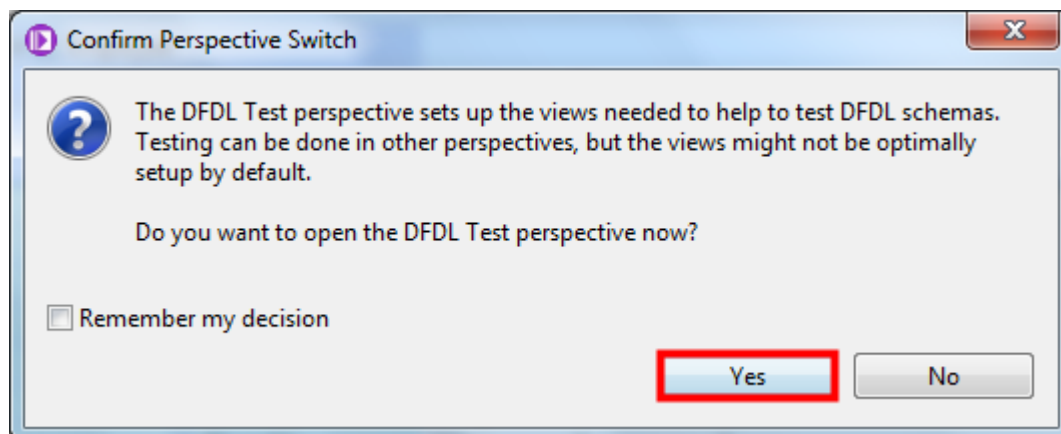
Runtime encoding options
Provide runtime values for properties which have been configured in the model to be dynamically set.
[More...](#)
Encoding (code page): UTF-8
Floating point format: IEEE Non-Extended
Byte order: ☐ Little endian ☒ Big endian

Runtime validation
☐ Validate data against schema [More...](#)

[Restore Defaults](#)

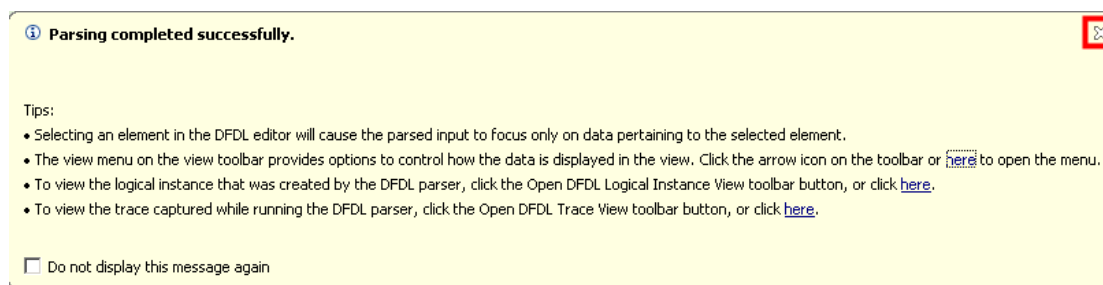
OK Cancel

__7. Press the **Yes** button to continue.



A pop-up dialog should display the results of the parsing test.

__8. Click the **X** to close the dialog box.



The DFDL Test perspective has several views:

- Parse: Allows you to parse a file using the message model in the editor.
- Trace: Has a detailed trace log of the testing activities. Very helpful to find the cause of errors.
- Logical Instance: Gives you a view of the parsed message tree.
- Serialize: Allows you to generate a file from the message model.

Employee.xsd

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee	sequence	1	1		
header	header	1	1		
record	record	1	unbounded		
EmpNo	string	1	1		value1
FirstName	string	1	1		value2

DFDL Test - Logical Instance

Data source: <From 'DFDL Test - Parse' view>

Message: Employee (/student/DFDL/workspace/MessageModellingLibrary/Emp)

Tree View XML View

Name	Type	Value
Employee		
header		
record		
record		
record		
record		
record		

DFDL Test - Parse

Status: Parsing completed: Tue Jul 23 03:31:08 EDT 2013

Input

Data: /MessageModellingLibrary/employees.csv Encoding (code page): UTF-8 Message: Employee (/MessageModellingLibrary/Employee.xsd)

Parsed Input

Characters

```

1 Company1 42 123 Street Winchester UK
2 "000010" "CHRISTINE" "I" "HAAS" "A00" "3978" "1995-01-01" "PRES" "18" "F" "1963-08-24" "152750.00
3 "000020" "MICHAEL" "L" "THOMPSON" "B01" "3476" "2003-10-10" "MANAGER" "18" "M" "1978-02-02" "94250.00
4 "000030" "SALLY" "A" "KWAN" "C01" "4738" "2005-04-05" "MANAGER" "20" "F" "1971-05-11" "98250.00
5 "000050" "JOHN" "B" "GEYER" "E01" "6789" "1979-08-17" "MANAGER" "16" "M" "1955-09-15" "80175.00
6 "000060" "IRVING" "F" "STERN" "D11" "6423" "2003-09-14" "MANAGER" "16" "M" "1975-07-07" "72250.00

```

The input text has been highlighted to differentiate the separators (",") that the parser has detected.

Parsed Input

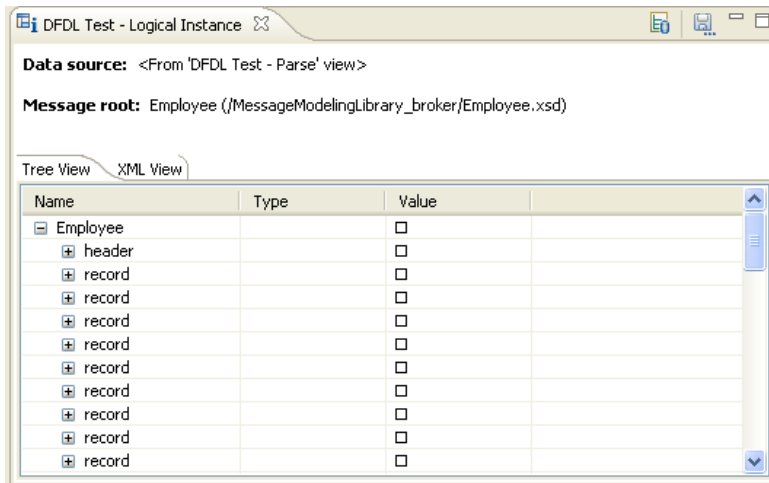
Characters

```

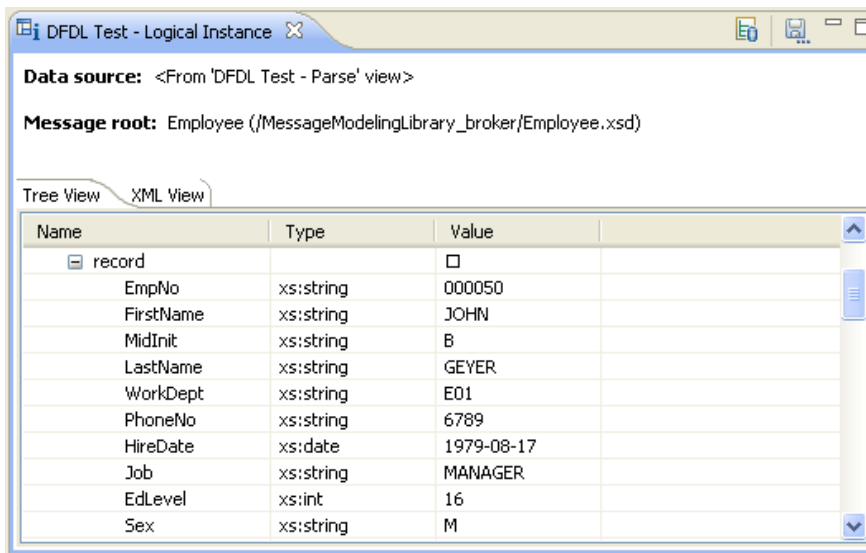
1 Company1 42 123 Street Winchester UK
2 "000010" "CHRISTINE" "I" "HAAS" "A00" "3978" "1995-01-01" "PRES" "18" "F" "1963-08-24" "152750.00
3 "000020" "MICHAEL" "L" "THOMPSON" "B01" "3476" "2003-10-10" "MANAGER" "18" "M" "1978-02-02" "94250.00
4 "000030" "SALLY" "A" "KWAN" "C01" "4738" "2005-04-05" "MANAGER" "20" "F" "1971-05-11" "98250.00
5 "000050" "JOHN" "B" "GEYER" "E01" "6789" "1979-08-17" "MANAGER" "16" "M" "1955-09-15" "80175.00
6 "000060" "IRVING" "F" "STERN" "D11" "6423" "2003-09-14" "MANAGER" "16" "M" "1975-07-07" "72250.00

```

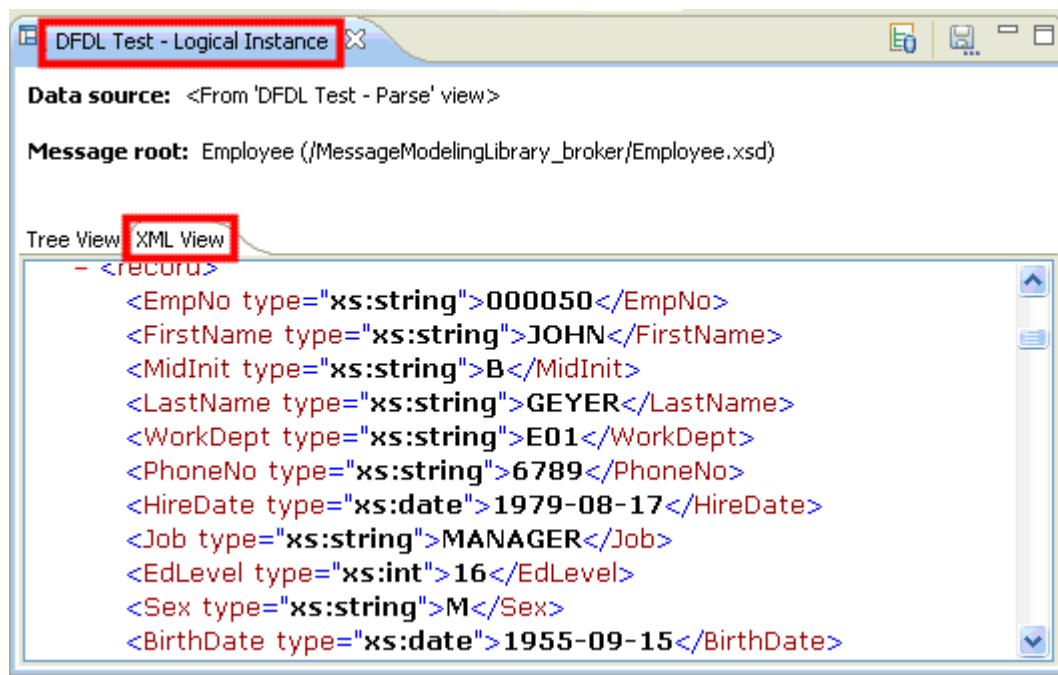
The Logical instance view is located in the top right corner. This shows the parsed message tree for the employees.csv file



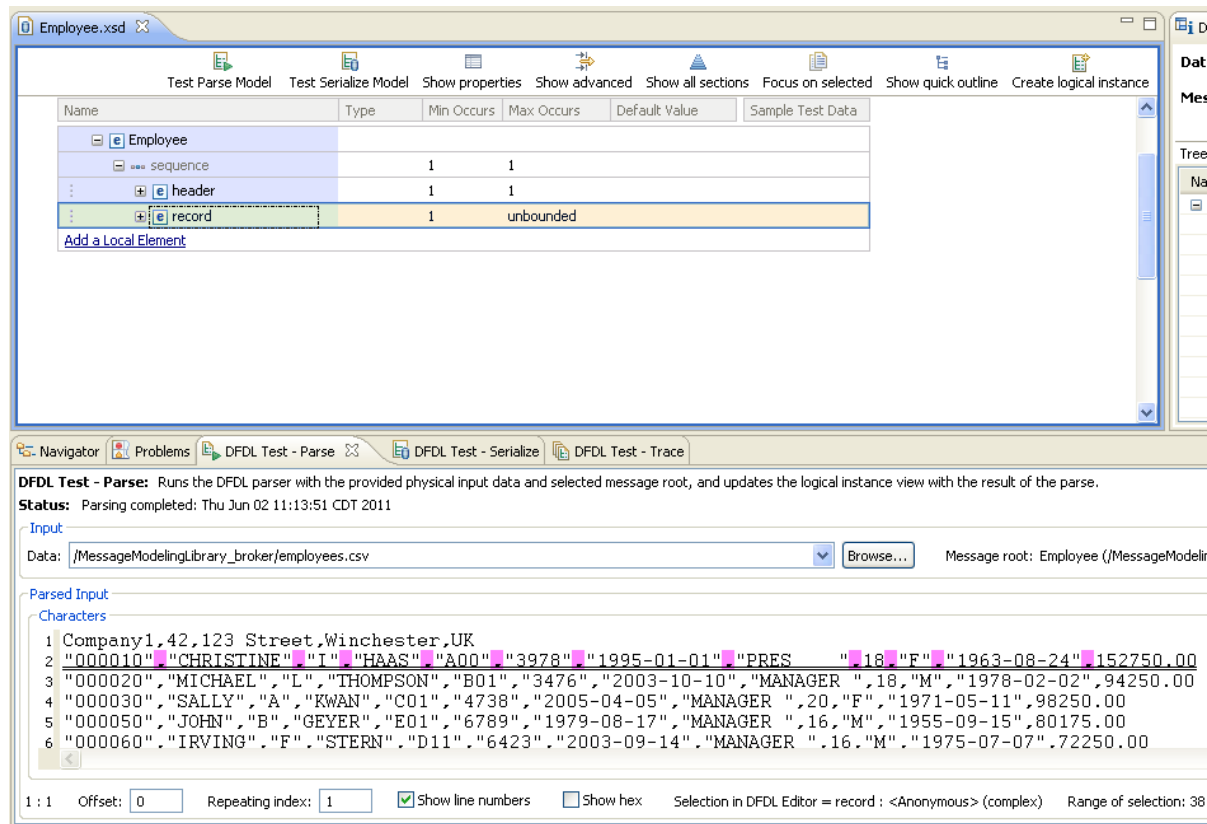
Expand a few records to check that the parsing was correct.



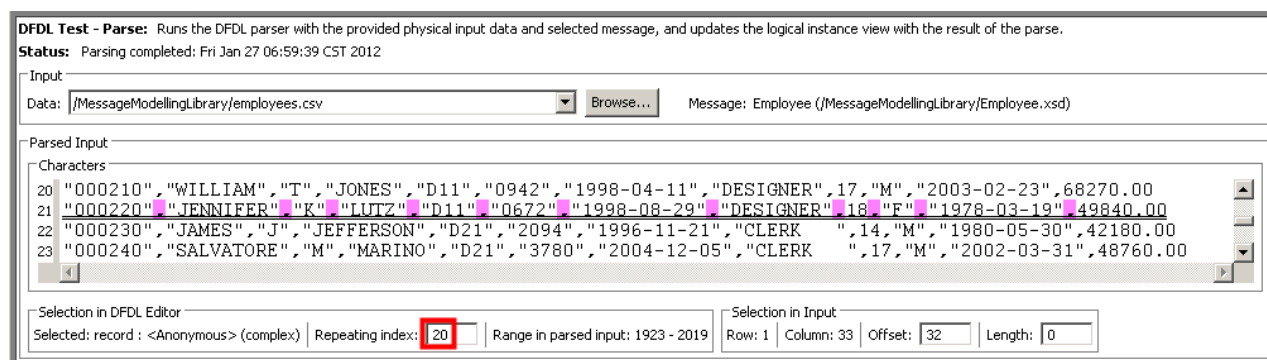
The **XML View** tab shows the same parsed records in XML format.



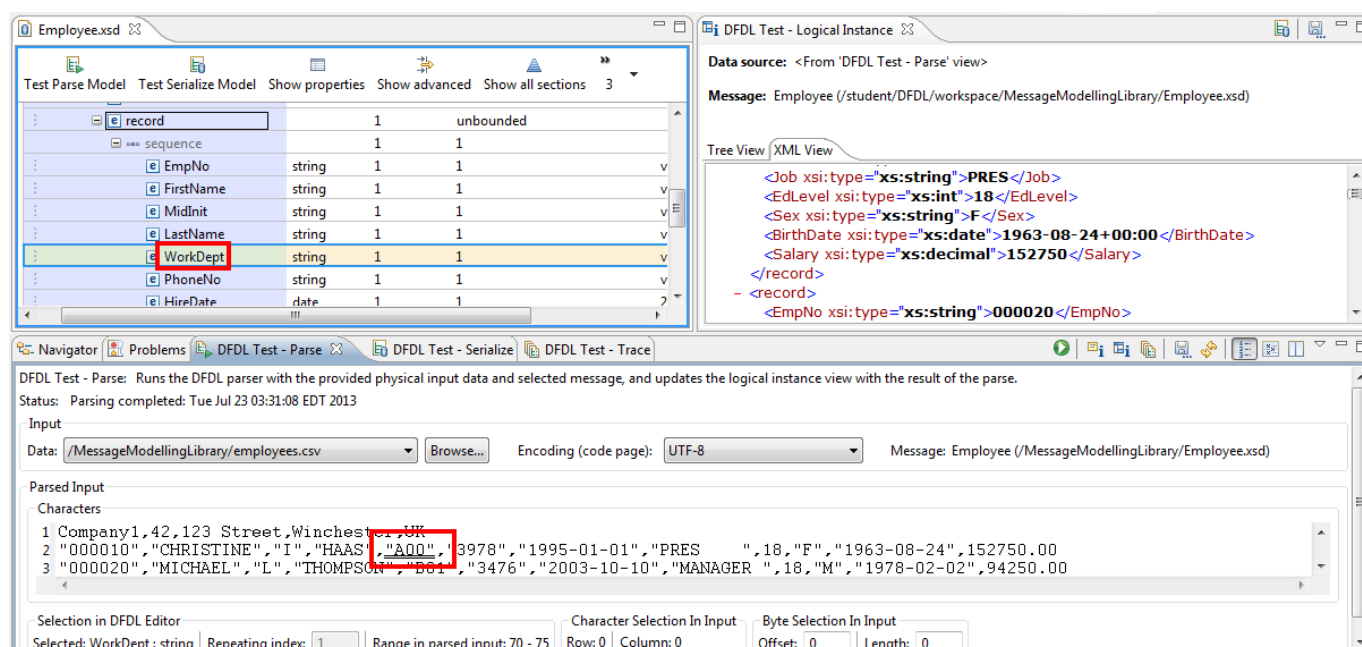
- __9. In the **Employee** message model, click the **Record** element. The first record in the input text will be underlined.



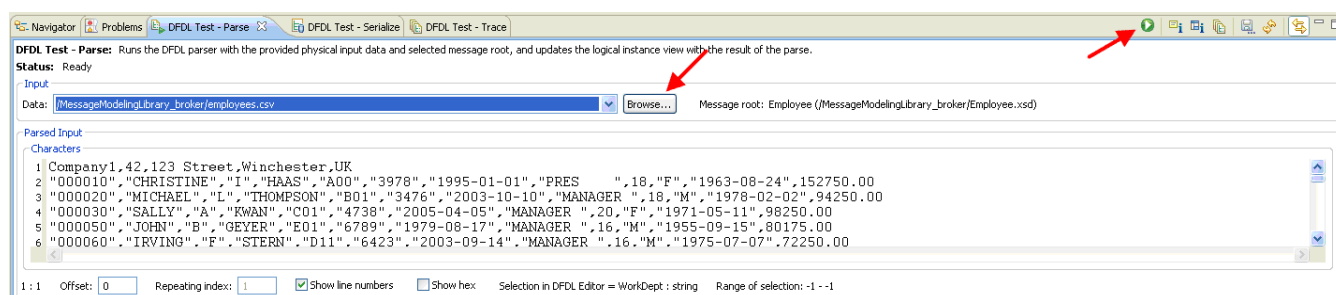
To go to another record in the input text, just change the **Repeating Index** number.



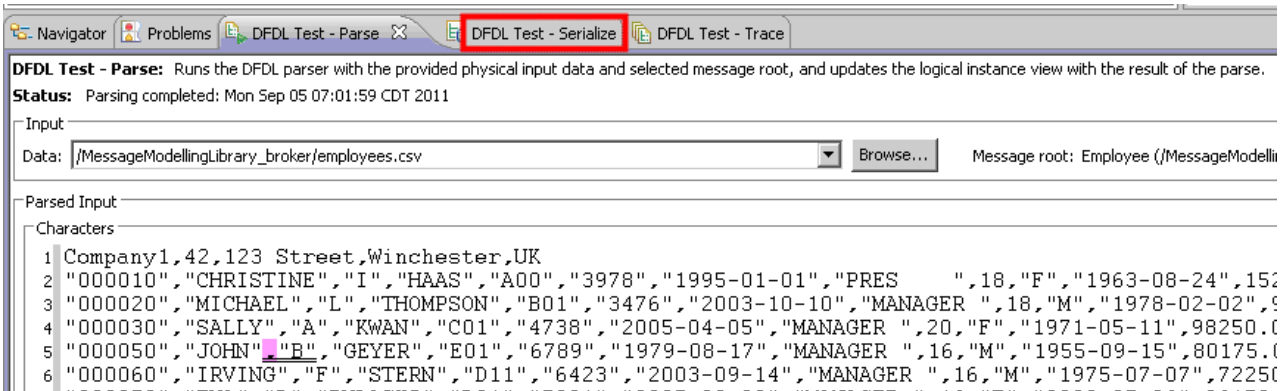
__10. Click any element in the message model to see where it's located in the input text.



The parsers can be run against others' input text by clicking the **Browse** button in the input section, selecting the file, and clicking the **Run Parser** button.

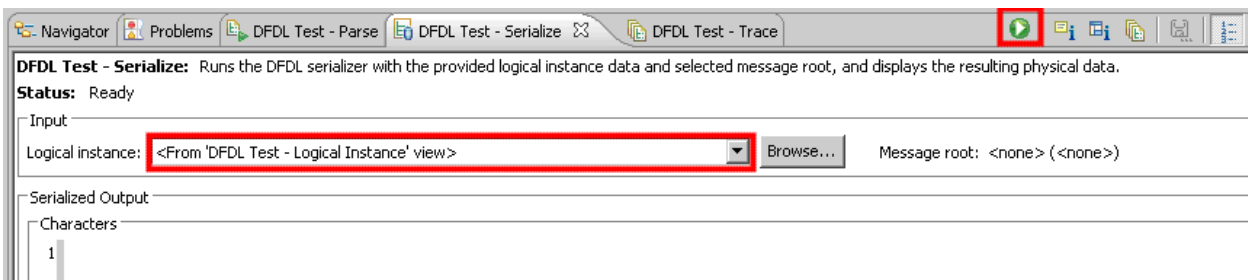


__11. Select the **DFDL Test – Serialize** tab

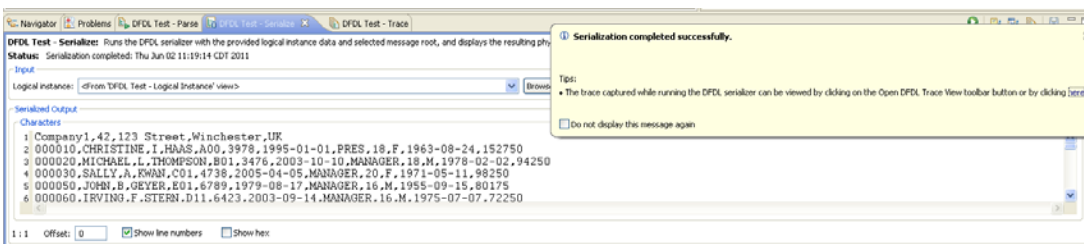


__12. Select **<From "DFDL Test – Logical Instance" view>** for the **Logical instance**, using the drop-down list.

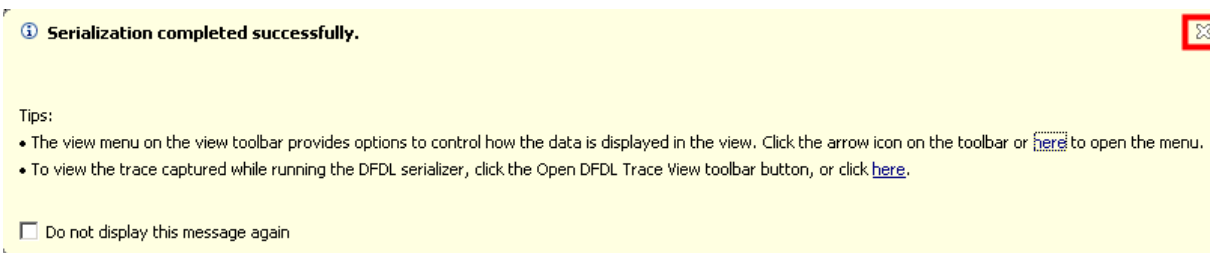
__13. Press the **Run Serializer** icon.



__14. The serializer creates a text file from the previously parsed message tree in memory.

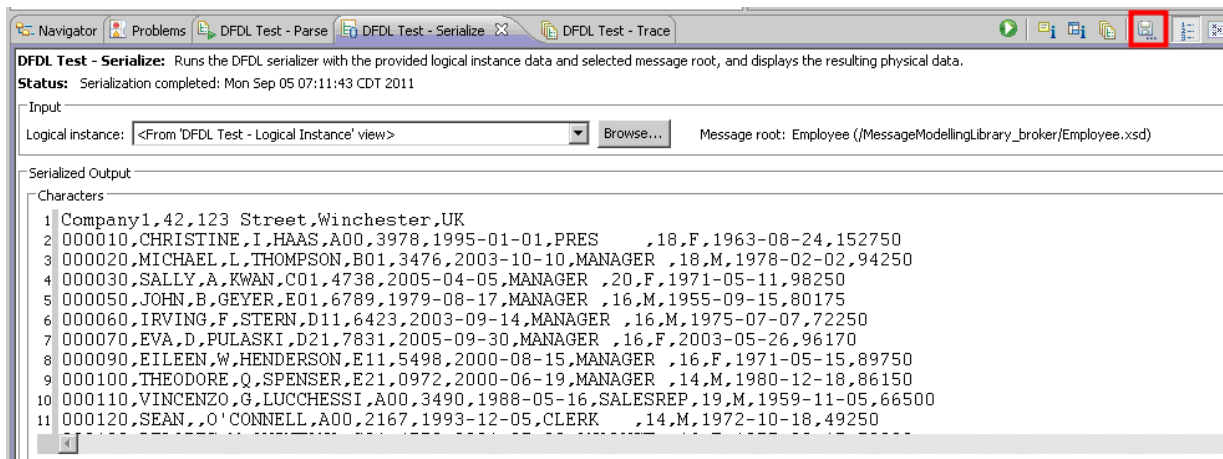


__15. Click the **X** icon to dismiss the results dialog.



The results of the serialization test should appear. The output will now be saved as a file.

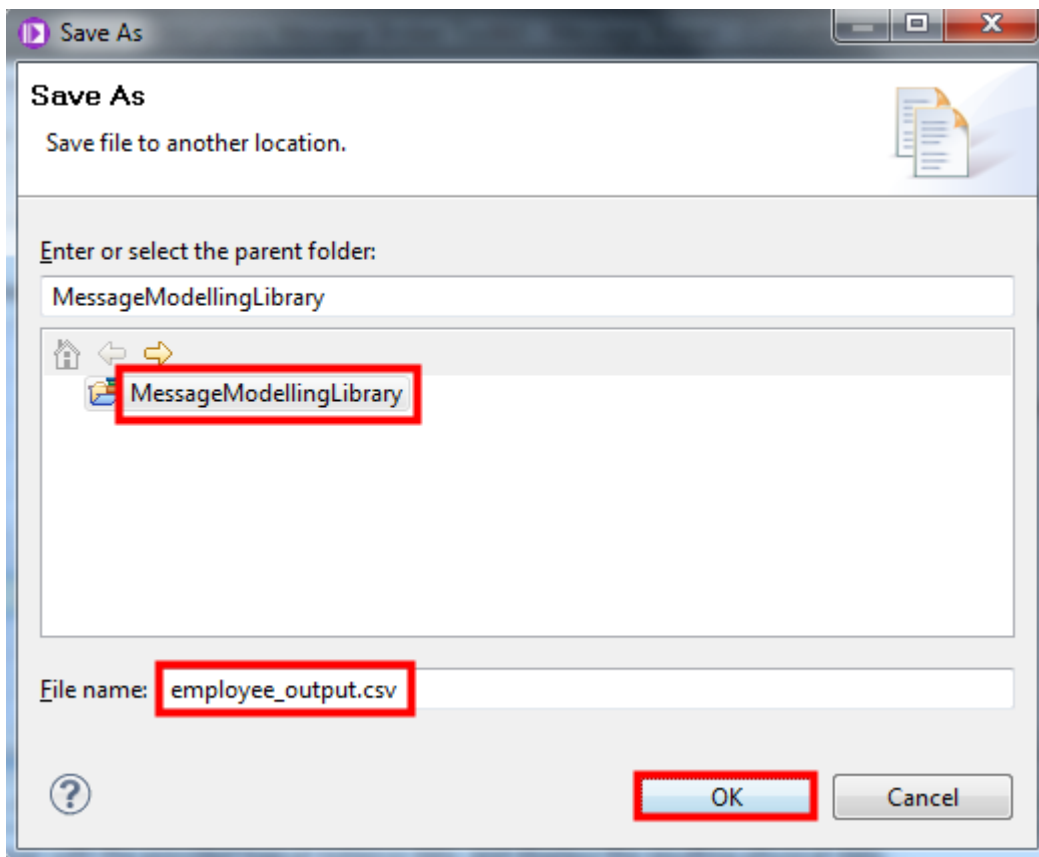
__16. Press the **Save to File** button (diskette icon).



__17. Select the **MessageModellingLibrary** as the parent folder.

__18. Enter **employee_output.csv** as the File name.

__19. Press the **OK** button.

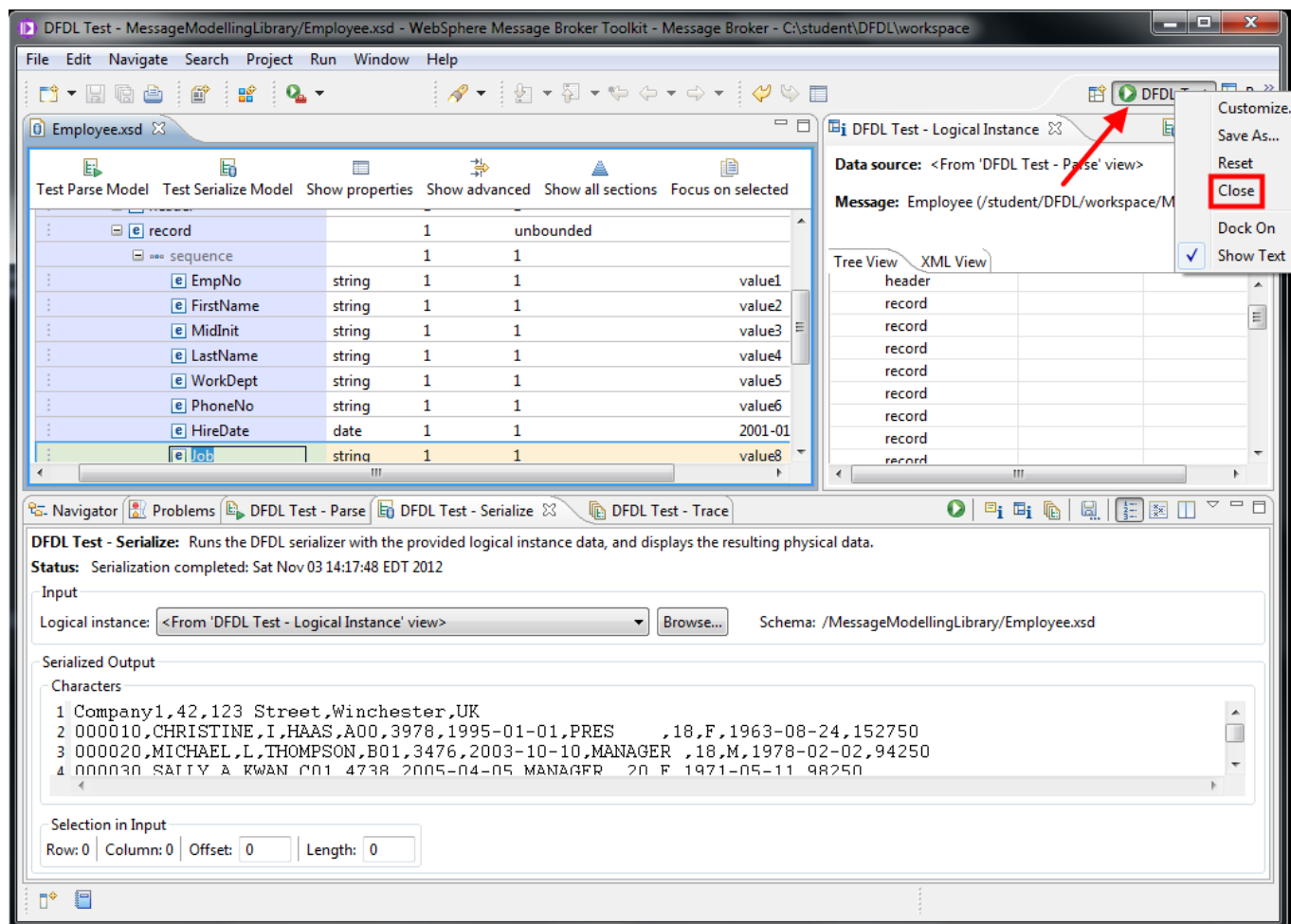


Close the **DFDL Test** perspective.

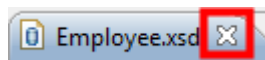
__20. Select the **DFDL Test** perspective tab in the upper right corner.

__21. Press the right mouse button.

__22. Select **Close** from the menu.



__23. Close the **Employee.xsd** tab.



This lab has provided a brief introduction to some of the new capabilities that have been provided with the DFDL parser. The DFDL domain should be the parser of choice for all non-XML data formats.

This is the end of Lab 8.

Appendix A. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have

been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
<i>System z</i>	<i>Tivoli</i>	<i>WebSphere</i>	<i>Workplace</i>	<i>System p</i>	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

NOTES



© Copyright IBM Corporation 2013.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, ibm.com and WebSphere are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.



Please Recycle
